Literature Survey

Student Name: Molly Hayward
Supervisor Name: Dr Noura Al-Moubayed
23/10/2019

Project title: An analysis of machine learning and deep learning techniques for the detection of sarcasm in text

I INTRODUCTION

A Problem Background

A form of irony, sarcasm is defined as the use of remarks that clearly mean the opposite of what they say, made in order to hurt someone's feelings or to criticize something in a humorous way [19]. This presents a fatal problem for classic sentiment analyzers, as text that appears to be positive at surface-level can instead convey an alternate negative meaning. Furthermore, failure to recognize sarcasm can be a human issue too, owing to the fact that its presence (or lack thereof) is considered rather subjective. González-Ibánez et al(2011) [7] showed that different humans do not classify statements as either sarcastic or non-sarcastic in the same way - for example, in their second study all three human judges agreed on a label less than 72% of the time. Truly, the existence of sarcasm can only be conclusively affirmed by the author. Additionally, developmental differences such as autism, as well as cultural and societal nuances cause variations in the way different people define and perceive sarcasm.

There exists an extensive body of literature covering machine learning and deep learning approaches to natural language processing problems, however in the application domain of sarcasm detection, they mostly display low accuracy. This could be due, in part, to the absence of a concrete, all-encompassing definition of sarcasm. N.B. this may also contribute to poor human accuracy in detecting a fundamentally human construct.

Despite the challenges in this domain, the ultimate aim of this project is to produce a tool that can detect sarcasm with a *high degree* of accuracy. In my endeavour to realise this aim, I will implement state-of-the-art word embedding and text classification techniques.

B Terms

NLP LSTM Irony []: Sarcasm []:

II THEMES

A Data collection

Sarcasm is typically used in informal discussions in person and online. Such data can be harvested from social networks in vast quantities, and many datasets are created in this way. For example, *Joshi et al. (2016)* [10] provides a succint comparison of many relevant datasets, of which the overwhelming majority consist of data extracted from social networks such as Reddit and Twitter. Much of the data is self-labelled by the author, either with markers such as #sarcasm in twitter data [4, 20, 21], or '/s' in reddit [11]. Collecting self-labelled means that the tweet's author has validated their intention of encorporating sarcasm. However, if they mislabel their tweet then this may introduce noise into the dataset. Additionally, most instances of sarcasm on twitter will not be indicated by the sarcasm hashtag, hence the set containing non-sarcastic tweets may also contain noise. Furthermore, the character limit of tweets is fixed at 140 - hence, essential context is often missing.

There are also publicly available datasets containing manually labelled data. The *News Headlines for Sarcasm Detection* dataset [15] consists of data originating from two news websites - one posting legitimate news and the other posting sarcastic synopses of current events. These headlines are written in a professional manner therefore spelling errors are less frequent. Furthermore, each headline is independent, unlike tweets which may be in reference to earlier posts. However, News headlines cover a smaller range of topics than twitter posts - hence, training a model in the narrow domain of news headlines will restrict the types of data that the model can generalise to. In addition, many datasets assume a balance between sarcastic and non-sarcastic samples, although this can lead to problems such as the *class inbalance problem*. Sarcasm occurs infrequently in conversation i.e. it is a minority class, and oversampling the minority class can lead to model overfitting.

B Data pre-processing

Data pre-processing is an important stage in most NLP tasks, with the potential to hinder or boost model performance. Often, data pre-processing is used to reduce noise and sparsity in the feature space, caused by inconsistent letter capitalization, varying use of punctuation and erroneous spelling. However, the suitability of each technique is highly dependent on the domain. For some datasets, certain types of pre-processing may be limiting, resulting in the removal of useful features. In this section, we consider the domain of sarcasm detection when discussing these pre-processing techniques.

On twitter, users can include hyperlinks to other websites in their tweets, and this can be a source of noise in the dataset. *Ptáček et al. (2014)* [20] collated a dataset of sarcastic and non-sarcastic tweets, whereby the presence of sarcasm is indicated by the marker #sarcasm, removing URLS and references to users, as well as hashtags. However, *Liebrecht et al. (2013)* [12] showed that data contained in hashtags can be used to indicate sarcasm, such as #not. Transforming text into lowercase is a common pre-processing strategy, useful for reducing sparsity caused by variations in letter capatiliztion e.g. 'Cat', 'cAt', and 'CAt' are all mapped to the same word - 'cat'. Similarly, removing punctuation and extending contractions (e.g. don't $\rightarrow$ do not) is commonplace. However, punctuation and capitalization can be used for emphasis, therefore removing these attributes may make the presence of sarcasm less obvious.

C Vectorization of textual data

Extracting meaningful data from large corpora is undoubtedly a complex task, however in order to do this successfully, we must first construct word vectors that capture semantic and syntactic relationships in a process known as vectorization. In this section, we examine this process with the aim of answering the question: how can we *best* vectorize text so as to encapsulate the important features of language?

Traditional Approaches – Perhaps the simplest vectorization approach is *Bag of Words (BOW)*, which encodes text as a single vector containing a count of the number of occurrences of each word in the snippet. *Term Frequency-Inverse document frequency (TF-IDF)* [23] extends the BOW model by providing each word with a score that reflects its level of importance based upon its frequency within the document. Predictably, vectors produced in this way do not preserve the context within which words are found, complicating the task of discerning their meaning. Hence, these approaches are unlikely to be suitable in the application domain of detecting sarcasm - a highly contextual phenomenon. The *Bag of N-grams* approach is a generalisation of Bag of Words, whereby instead of counting the number of occurrences of each unigram, we count the number of occurrences of each N-gram. This allows us to capture a small window of context around each word, however this results in a much larger and sparser feature set.

Word Embedding – First introduced in Mikolov et al (2013a) [14], *Word2Vec* describes a group of related models that can be used to produce high-quality vector representations of words - two such models are *Continuous Bag-of-Words* and *Continuous Skip-Gram*. It consists of a shallow (two-layer) neural network that takes a large corpus and produces a high-dimensional vector space. Unlike previous techniques, words that share a similar context tend to have collinear vectors, that is to say they are clustered together in the feature space. Consequently, Word2Vec is able to preserve the semantic relationships between words, even constructing analogies by composing vectors e.g. king - man + woman $\approx$ queen. Likewise, it captures syntactic regularities such as the singular to plural relationship e.g. cars - car $\approx$ apples - apple. In Word2Vec, intrinsic statistical properties of the corpus, which were key to earlier techniques, are neglected, therefore global patterns may be overlooked. To mitigate this, the *GloVe* [16] approach generates word embeddings by constructing an explicit word co-occurrence matrix for the entire corpus, such that the dot product of two word embeddings is equal to log of the number of times these words co-occur (within a defined window).

Despite their ability to preserve semantic relationships, Word2Vec and GloVe do not accommodate polysemy, which describes the co-existence of alternative meanings for the same word. In addition, they cannot generalise to words that were not specifically included in the training set. *Bojanowski et al (2017)* [3] describes a different vectorization technique used in Facebook's open-source fastText library. In the fastText approach, each word is represented as a bag of character n-grams which enables it to capture meaning for substrings such as prefixes and suffixes. In order to produce word embeddings for out-of-vocabulary tokens i.e. words not included in the training set, fastText composes vectors for the substrings that form the word. However, like Word2Vec and GloVe, FastText produces a single embedding for each word - hence, it cannot disambiguate polysemous words.

Contextualized Word Embedding – In pursuit of a more robust approach, we look to deep neural language models. Peters et al (2018) [17] introduced the *ELMo* model, and showed that the addition of ELMo to existing models can markedly improve the state-of-the-art in a number of NLP tasks. ELMo utilizes a bi-directional LSTM *(long short-term memory)* model, concatenating the left-to-right and right-to-left LSTM in order to produce deep contextualised word representations. Like fastText, they are character based, therefore robust representations can be constructed for out-of-vocabulary tokens. However, rather than 'looking-up' pre-computed embeddings, ELMo generates them dynamically. Hence, it can disambiguate the sense of a word given the context in which it was found.
Devlin et al. (2018) [5] introduced the *BERT* framework, a multi-layer bidirectional transformer model consisting of two main steps - *pre-training* and *fine-tuning*. During pre-training, unlabeled data is used to train the model on different tasks, providing some initial parameters that are tweaked in the fine-tuning stage (a procedure known as *transfer learning*). BERT uses a *masked language model* which "masks" 15% of the input tokens with the aim of predicting them based on their context. This allows BERT to leverage both left and right contexts simultaneously, unlike ELMO which uses shallow concatenation of separately trained left-to-right and right-to-left language models.

D Classification

Most approaches treat sarcasm detection as a binary classification problem, in which text is grouped into two categories - sarcastic and non-sarcastic. In the following text, I will compare methods used specifically in sarcasm detection research, as well as techniques that have seen success in other NLP classification tasks as they may have potential to perform well in this specific domain.

Traditional Classifiers – A number of simple linguistic approaches have been used in previous research into sarcasm detection. One such class of naive approaches is *rule-based*, where text is classified based on a set of linguistic rules. Maynard and Greenwood (2014) [13] proposed that the sentiment contained in hashtags can indicate the prescence of sarcasm in tweets, such as #notreally in the example "I love doing the washing-up #notreally". They used a rule-based approach to determine that if the sentiment of a tweet contradicts that of the hashtag, then this tweet is an example of sarcasm. In Riloff et al. (2013) [22], sarcasm is described as a contrast between positive sentiment and negative situation. This description is leveraged by Bharti et al (2015) [2], which presents two rule-based approaches to sarcasm detection. The first approach identifies sentiment bearing situation phrases, classifying the phrase as sarcastic if the negative situation is juxtaposed with a positive sentence. Rule-based techniques are fairly primitive when compared to their modern, deep learning counterparts. Especially when you consider that each ruleset must be generated manually and for each dataset and domain.

Machine-Learning Classifiers – Using labelled training data, a machine-learning algorithm can learn patterns that are associated with each category of data. This allows it to classify unseen text into these categories without the need for a static linguistic rule set. If the training data is high-quality and plentiful, the model can begin to make accurate predictions. There are a few general categories of machine-learning classifiers, including Naive Bayes, Support Vector machines (SVMs) and logistic regression based classifiers. Although, not all approaches fit neatly into these categories.

For example, Tsur et al. (2010) [24] used a semi-supervised algorithm, *SASI*, in order to detect sarcasm in online product reviews. Their dataset contained 66000 Amazon reviews, each annotated by three humans as a result of crowdsourcing. They extracted syntactic and pattern-based features from the corpus, then used a classifcation strategy similar to k-nearest neighbor (kNN), whereby they assigned a score to each feature vector in the test set by computing the weighted average of the k-closest neighbors in the training set N.B. neighbours are vectors that share at least one pattern feature. This model had relative success - achieving a precision of 77% and recall of 83.1% on newly discovered sentences.

Naive Bayes classifiers are supervised algorithms that use bayes theorem to make predictions. Reyes et al. (2013) [21] used naive bayes and decision trees algorithm in order to detect irony which is closely related to sarcasm. They experimented with balanced and unbalanced distributions (25% ironic tweets and 75% other), achieving an F-score of 0.72 on the balanced distribution, dropping to 0.53 for the inbalanced distribution. In a similar vein, Barbieri et al. (2014) [1] used random forest and decision tree classifiers, also for the detection of irony. They used six types of features in order to represent tweets, and recorded results over three categories of training data - education, humor and politics.

Support Vector Machines (SVM), can be effective when smaller amounts of training data are available, however they require more computational resources than naive bayes. SVMs use kernel functions to draw hyperplanes dividing a space into subspaces. González-Ibánez et al(2011) [7] used two classifiers, support vector machine with sequential minimal optimization, as well as logistic regression. Their best result was an accuracy of 0.65, achieved with the combination of support vector machine with sequential miniminal optimization and unigrams. *Ptáček et al. (2014)* [20] also used support vector machine and Maximum Entropy classifiers. They also performed classification on balanced and inbalanced distributions.

Deep-Learning Classifiers – Deep neural networks (DNNs) are increasingly being used in text classification tasks [18, 26]. Two common DNN architectures are Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). They typically require a lot more training data than machine learning algorithms, but have been shown to produce state-of-the-art results in several domains.

Recurrent Neural Networks are good at processing sequential, or time-series, information such as text. This is due to the fact that RNNs have a 'memory', in that the input to the current step is the output from the previous step. However, they suffer from the vanishing gradient problem. This is where gradient values get smaller as it backtracks through lower layers, making it harder for the network to update weights and causing calculations to take longer. Recurrent Neural Networks are not very good at remembering long term dependencies, therefore in this instance we can use *Long short-term memory (LSTM)* models [9] instead which are more effective at preserving long term dependencies.

Zhang et al. (2015) [26] explored the used of character-level convolutional neural networks for text classification. This network consists of 6 convolutional layers and 3 fully-connected layers. It performed better on raw texts such as Amazon reviews than

Poria et al. (2017) describes a convolutional neural network (CNN) for sarcasm detection.

They typically require a lot more training data than machine learning algorithms, however unlike machine learning algorithms, they get more accurate the more data they are fed without being capped at a certain threshold.

CNN allows us to extract higher-level features, has applications in sentiment analysis. Collobert and Weston were among the first to apply CNN-based frameworks to NLP tasks RNNs are probably better

Gated Recurrent Units (GRUs)

LTSMs and GRUs introduced to overcome this

An LSTM consists of three gates that update and control cell states - the forget gate, inp

One particularly interesting technique is the *Hierarchical Attention Network (HAN)* defined in Yang et al. (2016) [25] - an attention-based LSTM. The intuition is that certain words and sentences in a document contribute more to its overall meaning, and this is highly context dependent. They included one attention mechanism at word-level and another at sentence-level, and this allowed them to determine which words and phrases correlate more to certain labels. Using a similar approach in the domain of sarcasm may allow me to highlight the attention-words that are strongly influence the level of sarcasm. A similar technique was applied in *Ghosh et al. 2018)* [6] where they concluded that emoticons such as ':)' and interjections e.g. "ah", "hmm" correspond with highly weighted sentences.

E Model Performance Evaluation

Lastly, evaluating the successes and failures of a trained model is a critical step. A simple approach is to use accuracy which refers to the proportion of data that is correctly labelled as either sarcastic or non-sarcastic. However, sarcasm is a minority class therefore on an unbalanced dataset we could achieve high accuracy by simply labelling every statement as non-sarcastic. In an attempt to mitigate against this, I will instead form a conclusion based on the F_1 score i.e. the harmonic mean of precision and recall, where scores range from 0 (worst) to 1 (best). This metric has faced some criticism for giving equal weight to both precision and recall [8], therefore I will consider both measures separately, as well as in combination.

$$\text{precision} = \frac{\text{true positives}}{\text{true positives + false positives}} \qquad \text{recall} = \frac{\text{true positives}}{\text{true positives + false negatives}}$$

In the domain of sarcasm detection, precision refers to the proportion of the classified-sarcastic data that is *truly sarcastic* i.e. how many of the positives are true positives, and recall describes the proportion of the truly sarcastic data in the corpus that is *classified* as such i.e. how many true positives are labelled as positives.

References

[1] Francesco Barbieri and Horacio Saggion. Modelling irony in twitter. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 56–64, 2014.

[2] Santosh Kumar Bharti, Korra Sathya Babu, and Sanjay Kumar Jena. Parsing-based sarcasm sentiment recognition in twitter data. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 1373–1380. ACM, 2015.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[4] 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*, pages 107–116. Association for Computational Linguistics, 2010.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] Debanjan Ghosh, Alexander R Fabbri, and Smaranda Muresan. Sarcasm analysis using conversation context. *Computational Linguistics*, 44(4):755–792, 2018.

[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] David Hand and Peter Christen. A note on using the f-measure for evaluating record linkage algorithms. *Statistics and Computing*, 28(3):539–547, 2018.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

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

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

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

[13] DG Maynard and Mark A Greenwood. Who cares about sarcastic tweets? investigating the impact of sarcasm on sentiment analysis. In *LREC 2014 Proceedings*. ELRA, 2014.

[14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[15] Rishabh Misra and Prahal Arora. Sarcasm detection using hybrid neural network. *arXiv preprint arXiv:1908.07414*, 2019.

[16] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[17] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[18] Soujanya Poria, Erik Cambria, Devamanyu Hazarika, and Prateek Vij. A deeper look into sarcastic tweets using deep convolutional neural networks. *arXiv preprint arXiv:1610.08815*, 2016.

[19] Cambridge University Press. Sarcasm: meaning in the cambridge english dictionary. `https://dictionary.cambridge.org/dictionary/english/sarcasm`, 2019. Last accessed: 23 Oct 2019.

[20] Tomáš Ptáček, Ivan Habernal, and Jun Hong. Sarcasm detection on czech and english twitter. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 213–223, 2014.

[21] Antonio Reyes, Paolo Rosso, and Tony Veale. A multidimensional approach for detecting irony in twitter. *Language resources and evaluation*, 47(1):239–268, 2013.

[22] 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 *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 704–714, 2013.

[23] Stephen E Robertson and K Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information science*, 27(3):129–146, 1976.

[24] Oren Tsur, Dmitry Davidov, and Ari Rappoport. Icwsm—a great catchy name: Semi-supervised recognition of sarcastic sentences in online product reviews. In *Fourth International AAAI Conference on Weblogs and Social Media*, 2010.

[25] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

[26] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.