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 [10]. 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) [3] 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

II THEMES

A Data pre-processing

Data pre-processing is the first stage in most NLP tasks, and has 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 factors such as inconsistent letter capitalization, varying use of punctuation and erroneous spelling. However, the suitability of each technique is highly dependent on the domain. Under some circumstances, 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.

Often, data pre-processing is used to reduce noise and sparsity in the feature space, caused by factors such as inconsistent letter capitalization, varying use of punctuation and erroneous spelling.

For example, by transforming text into lowercase and removing hyphens, caN't, Can't, CANt (and other variations) are all mapped to the same word - cant.

Lowercasing – 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 all become the same word - cat.
Sarcasm is typically used in informal discussions both online and in person. Such data can be harvested from social networks in vast quantities, and many datasets are created in this way. For example, *Joshi et al. (2017)* [5] 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.

For example, punctuation removal is a common form of data cleaning.

Given that punctuation such as "!" can indicate emphasis,

Some data cleaning techniques consider removal of punctuation, however

Kreuz and Caucci (2007) analyzed the effect of lexical influences, such as interjections and punctuation, on the perception of sarcasm.

Oftentimes, these datasets are balanced, despite the fact that sarcasm occurs infrequently.

Spelling correction

Data cleaning, annotation, normalization, are all forms of data pre-processing.

Tokenization, data cleaning and normalization are all data pre-processing steps

For example, stemming and lemmatization can both be used in normalization, as well as case transformation.

There are numerous data pre-processing techniques used in a variety of NLP tasks.

In reality, sarcasm occurs infrequently, it doesn't crop up often.

Riloff et al (2013) interprets sarcasm as a contrast between positive sentiment and negative situation.

Sarcasm is typically used in informal conversations

The character limit of tweets is fixed at 140 - hence, essential context is often missing.

A number of datasets have compiled statements from Reddit [6, 15]

Removing stopwords and other aspect Some techniques word rem

Some applications advocate for the use of stemming, such as search applications, e.g. searching for 'deep learn' should prodce similar results to 'deep learning'.

We could perform topic modelling on a general social media dataset Based on this, we could see which sorts of topics are more likely to indicate sarcasm?

Social networks are a huge database of informal language.

On the one hand, collecting self-labelled data in this way 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.

Most instances of sarcasm on twitter will not be indicated by the sarcasm hashtag, therefore the set of non-sarcastic tweets may also contain noise in the form of un-marked sarcastic statements.

In this dataset, characters are restricted to ASCII, therefore

One potential option is to include a fixed word dictionary containing positive and negative emoticons and include this as an additional feature.

Similarly, the Reddit dataset [6] This is an unbalanced, self-annotated dataset

Reddit users terminate a sarcastic statement with the "/s" marker.

They collected

Riloff et al (2013) [12] collected

Conversations on social media tend to be informal, however this will introduce noise into the dataset in the form of erroneous spelling,

I expect to encounter some difficuties when it comes to processing informal and erroneous spelling. Additionally, in a dataset of tweets, text is fairly short and could be lacking essential context. Hence, it may inadequate just to analyze the text independently. Some contextual information, such as the number of followers author of the tweet, may be useful to add more meaning.

The previous sections have highlighted the ambiguities within sarcasm. Riloff et al (),

This method is not without its flaws. Some people may incorrectly label their tweets as sarcastic, leading to noise in the dataset. Similarly, the non-sarcastic tweets are a subset of all other tweets that are not labelled with sarcasm and it goes without saying that a small percentage of these tweets will in fact be sarcastic - another source of noise. Likewise, numerous studies have shown that humans disagree over labelling statements as sarcastic.

B 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)* [13] 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) [7], *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* [8] 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)* [1] 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) [9] 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) [2] 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.

C 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.

Tsur et al(2010) [14] used a semi-supervised algorithm, *SASI*, in order to detect sarcasm in online product reviews. They used a dataset containing 66000 Amazon reviews, each annotated by three humans as a result of crowdsourcing. They begin with a small set of seed sentences,

First, 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.

Traditional Classifiers – Standard linguistic approaches One such naive approach is rule-based, where texts is grouped based on a set of linguistic rules. These rules may be refined for a specific domain, or simply generalised to any kind of text. Generating these rules can be time consuming and requires in-depth knowledge of the domain.

Wallace et al(2014) showed that humans require access to context in order to judge sarcastic tone, hence it may be naive to assume that machines can perform well on this task without it.

have seen success in other NLP classification tasks, such as ... In the domain of sarcasm detection,

A growing trend appears to be extraction

Joshi et al(2016) explores the usefulness of word-embedding based features in sarcasm detection.

Riloff et al(2013) presents a bootstrapping algorithm for detecting a specific style of sarcasm - contrasting a negative situation with a positive sentiment. This is an oversimplification of sarcasm, hence it is unlikely to capture less explicit formulations.

Machine-Learning Classifiers – Using labelled training data, a machine-learning algorithm can learn the hallmarks associated with each category. This allows it to classify text without the need for a static linguistic rule set. First, we need to perform feature extraction on the input text to produce a set of features. We then combine these features with tags (e.gpositive or negative in sentiment analysis) to produce a classification model. If the data is high-quality and plentiful, the model can begin to make accurate predictions. - Naive Bayes — A family of statistical algorithms, based on bayes theorem. MNB (Multinomial naive bayes allows us to make accurate predictions even with a small amount of training data) Quick description of Naive Bayes and the results of a few papers that have used it - Support Vector Machines (SVM), doesn't need a lot of training data but needs more computational resources than naive bayes - can achieve more accurate results. It draws a line / hyperplane diving a space into two subspaces (one containing vectors that belong to a group, the other containing the rest)

[3] used two classifiers, support vector machine with sequential minimal optimization, as well as logistic regression

[11] also used support vector machine classifiers

- Logistic Regression Quick description of SVM and the results of a few papers that have used it Machine learning algorithms reach a certain threshold where adding more training data doesn't improve their accuracy.

Ptacek et al() also form their english dataset using the same approach.

Ghosh et al2015 used SVMs

Some studies have instead used amazon product reviews, for example Tsur et al(2010) analyzed sentences from a dataset of 5.9 million tweets and 66000 product reviews from Amazon. They used crowdsourcing to annotate the corpus

Deep-Learning Classifiers – Two common architectures used in text classification are Convolutional Neural Networks and Recurrent Neural Networks. 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

RNNs are specialized neural-based approaches, good at processing sequential / time-series information (i.e. text) However, they suffer from the vanishing gradient problem

LTSMs and GRUs introduced to overcome this

An LSTM consists of three gates,

D 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 [4], 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] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[2] 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.

[3] 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.

[4] 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.

[5] Aditya Joshi, Pushpak Bhattacharyya, and Mark J Carman. Automatic sarcasm detection: A survey. *ACM Computing Surveys (CSUR)*, 50(5):73, 2017.

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

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

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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)*, pages 512–516, 2014.