An Analysis of Machine Learning and Deep Learning Techniques for Detecting Sarcasm in Online Text

Student Name: Molly Hayward

Supervisor Name: Dr Noura Al-Moubayed

Submitted as part of the degree of BSc Computer Science to the

Board of Examiners in the Department of Computer Sciences, Durham University

Abstract —

Context / Background – Sarcasm detection is a relatively new research topic that has recently gained interest due to the increased use of sentiment analysis and social media analytics. Sarcasm is a unique form of language that cannot be interpreted at surface-level, therfore detecting sarcasm proves a significant challenge for traditional sentiment analysers and humans alike. This highlights the scope for innovative machine learning and deep learning solutions to this ill-structured problem.

Aims – The ultimate aim of this project is to develop a tool that can detect sarcasm in online text from multiple domains with a *high degree* of accuracy. We aim to determine whether deep neural network architectures are more effective than machine learning classifiers for this task, and to what extent manually extracted sentiment features can be used to improve machine learning classification. Finally, we aim to highlight tokens that correlate more to sarcastic labels.

Method – We gather labelled datasets from online sources containing Amazon reviews, Twitter data and News headlines. Multiple vectorisation approaches are used to transform text into numerical representations, including a state-of-the-art contextualised embedding approach (ElMo). These vectors are used to train various machine learning and deep learning classifiers to detect the presence of sarcasm. In addition, an attention mechanism is incorporated to localise the presence of sarcasm within a passage of text.

Results – Through extensive experimentation, we show that

Conclusions – In summary, we conclude that

Keywords — Machine learning, Deep learning, Sarcasm Detection, Sentiment, Classification

INTRODUCTION

Sarcasm is a complex linguistic phenomenon, which when present in text indicates that the literal interpretation differs from the implied meaning. Sarcasm is prevalent in online user-generated content, and poses a significant challenge within the field of natural language processing, specifically within sentiment analysis, as it inverts the sentiment-polarity of a positive or negative utterance. Sentiment analysis is the task of deducing the sentiment polarity of text - typically whether the author is in favour of, or against, a certain subject. It is increasingly common for organisations to use sentiment analysis in order to gauge public opinion on their products and services; however, classic sentiment analysers cannot deduce the implicit meaning of sarcastic text and will wrongly classify the author's opinion. Hence, any tool that strives to accurately determine the meaning of user-generated text must be capable of detecting sarcasm. The phenonmenon of sentiment incongruity often lies at the heart of sarcasm, therefore advancements in

automatic sarcasm detection research have the potential to vastly improve the task of sentiment analysis in opinion mining.

A Problem Background

As sarcasm is multi-faceted, this makes its detection a unique and interesting challenge. It can be both explicit and implicit; oftentimes, contextual cues are more powerful indicators of sarcasm than the words themselves. However we often lose this context, and hence the sarcastic undertones, in transcription. Consider the scenario where a person is congratulated on their hard work, despite the obviousness of them having not worked hard at all. Or perhaps a customer thanking a waiter for the delicious food, even though they sent it back to the kitchen. In isolation, the speech is not enough to convey the sarcastic intent. Furthermore, even humans struggle to consistently recognise sarcasm; due in part to the lack of an all-encompassing, universal definition . In a 2011 study, González-Ibánez et al. [5] found low agreement rates between human annotators when classifying statements as either sarcastic or non-sarcastic, and in their second study, three annotators unanimously 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.

These factors make sarcasm detection an extremely complex task for both humans and computers. Despite this, *most* humans can recognise sarcastic intent *most* of the time. We strive to replicate this performance, or even perhaps improve upon it, in order to move towards a more concrete definition of what *makes* a statement sarcastic. This highlights the scope for automating this process, and the need for an innovative solution.

Early rule-based approaches introduced the idea that the sentiment contained within text can be indicative of sarcasm, although exploring these sentiment features using more complex machine learning and deep learning architectures is a novel approach to the problem. The first deep learning approach to sarcasm detection was used in *Poria et al. (2017)* [17] to produce state of the art results.

B Research Questions and Objectives

The **research questions** guiding this project are as follows –

- Can handcrafted sentiment features improve sarcasm predictions?

- Do deep learning techniques perform better than machine learning approaches?

- Can a custom model identify which linguistic cues correlate more to sarcastic labels?

The following objectives are designed to address these specific research questions, and are divided into three categories depending on their priority level and difficulty.

The **minimum** objectives of this project are to evaluate, compare and clean high-quality datasets, as well as to employ machine learning models and evaluate the effectiveness of these architectures on our chosen datasets. We found that a logistic regression model trained on ElMo vectors performed best for all three datasets, achieving F_1 scores of 84.6%, 81.3% and _%.

The **intermediate** objectives of this project are to experiment with and evaluate deep learning

[1] www.dictionary.cambridge.org/dictionary/english/sarcasm

models on the chosen datasets, and to determine which is the best performing solution. Additionally, we evaluate the model against unseen data. This is achieved by ...

The **advanced** objectives of this project are to implement an attention-based neural network in order to identify words that correlate more to sarcastic labels, in order to produce a visualisation of attention words. We implement a custom attention layer using keras, and apply our trained model to a number of sentences to illustrate its effectiveness.

This is achieved by...

RELATED WORK

In this section, we report on the architectural design of solutions used specifically in sarcasm detection research, as well as techniques that have seen success in other natural language processing (NLP) classification tasks. Oftentimes, sarcasm detection is treated as a binary classification problem, in which text is grouped into two categories - sarcastic and non-sarcastic. It is otherwise treated as a multi-class problem where the extent to which a statement is sarcastic is ranked on a discrete scale, e.g. 1 to 5. In similar studies, the terms irony and sarcasm are used synonymously [23], although their definitions differ slightly (*table 1*). Both definitions capture the humourous intent in both sarcasm and irony, however sarcasm includes the potential for mockery and criticism - often present in opinion-based content such as product reviews.

Table 1: Definitions of Irony and Sarcasm

Term	Definition
Irony	A subtle form of humour which involves saying things that you do not mean. [1]
Sarcasm	Speech or writing which actually means the opposite of what it seems to say, usually intended to mock or insult someone. [2]

A Rule-based Classifiers

A number of naïve approaches have been used in previous research, where text is classified based on a set of linguistic rules. Bharti et al. (2015) [2] proposed two rule-based approaches - one approach (IWS) classifies tweets that begin with injerjections, such as *"wow"* and *"yay"*, as sarcastic, and the other uses a parsing based lexical generation algorithm (PBLGA) to recognise sentiment-bearing situation phrases juxtaposed with contrasting sentiment phrases. When tested on 1500 tweets containing #sarcasm, the IWS and PBLGA approaches yielded F_1 scores of 0.90 and 0.84, respectively. An earlier example of a rule-based approach is displayed in Maynard and Greenwood (2014) [12]. They proposed that the sentiment contained in hashtags can indicate the presence of sarcasm in tweets, such as #notreally in the example *"I love doing the washing-up #notreally"*. They used a rule-based approach where tweets are labelled as sarcastic if their sentiment contradicts the sentiment of the hashtag, reporting an F_1 score of 0.91.

Rule-based classifiers can perform well if their rule-sets are tailored to specific datasets - however, they cannot generalise effectively to data that is structurally different, hence they are considered primitive when compared to their modern, deep learning counterparts.

B Machine-Learning Classifiers

Machine learning algorithms do not rely upon a static linguistic rule-set. Instead, they learn the underlying patterns associated with each class, allowing them to separate unseen data into predetermined categories. There are a number of commonly used model architectures, including Naïve Bayes classifiers, Support Vector Machines (SVMs) and Logistic Regression classifiers, although not all approaches fit neatly into these categories.

An early appproach, Tsur et al. (2010) [23], proposed a novel semi-supervised algorithm, *SASI*, for sarcasm identification; trained on a small balanced seed of 160 reviews (80 sarcastic and 80 non-sarcastic) from a corpus of 66000 human-annotated Amazon product reviews. In

[1] https://www.collinsdictionary.com/dictionary/english/irony

[2] https://www.collinsdictionary.com/dictionary/english/sarcasm

order to mimic the ambiguous and spectral nature of sarcasm, a discrete score between 1 (non-sarcastic) and 5 (definitely sarcastic) is assigned to each sentence; scores of 3 and higher indicate sarcasm. Syntactic and pattern-based features are extracted and fed to a classifier which utilises a strategy similar to K-Nearest Neighbours (kNN), whereby a sarcasm score is assigned to an unseen vector based upon the weighted average of the k closest vectors from the training set. They reported an F_1 score of 82.7% on the binary classification task.

In Riloff et al. (2013) [20], sarcasm is described as a contrast between positive sentiment e.g. *"i love"*, in reference to a negative situation, e.g. *"being ignored"*. They use a bootstrapping algorithm to extract positive verb phrases and negative situation phrases from a Twitter corpus and used these to train a hybrid model consisting of an SVM and a contrast method which classifies text as sarcastic if a postive sentiment phrase precedes a negative situation. This yielded an F_1 score of 51%.

Reyes et al. (2013) [19] used a naïve 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.

González-Ibánez et al. (2011) [5] used a Support Vector Machine (SVM) with sequential minimal optimisation, as well as Logistic Regression, on lexical (e.g. unigrams) and pragmatic (e.g. emoticons) features. They achieved 71% accuracy using an SVM trained on unigrams.

Ptáček et al. (2014) [18] also used SVM and Maximum Entropy classifiers. They also performed classification on balanced and inbalanced distributions.

C Deep-Learning Classifiers

Deep neural networks (DNNs) are increasingly being used in text classification tasks [17, 26]. Though there are many forms of DNN, two commonly used architectures are Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). Oftentimes, deep neural networks require a lot more training data than machine learning algorithms, but have been shown to produce state-of-the-art results in several domains. If the training data is high-quality and plentiful, then the model can begin to make accurate predictions.

A similar technique was applied in *Ghosh et al. 2018)* [4] where they concluded that emoticons such as ':)' and interjections e.g. "ah", "hmm" correspond with highly weighted sentences. Gated Recurrent Units (GRUs) are another type of RNN used to overcome the vanishing gradient problem. *Zhang et al. (2016)* [25] used a bidirectional gated RNN to capture local information in tweets, as well as a pooling neural network to extract context from historic tweets, achieving 78.55% accuracy.

Convolutional Neural Networks allow us to extract higher-level features, and they are based on the mathematical operation of convolution. *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. They found that it showed better peformance on raw texts such as an Amazon product review corpus.

In *Poria et al. (2017)* [17], cited as the first study to use deep learning for sarcasm detection.

CNNs have been previously used in sarcasm detection. For example, *Poria et al. (2017)* [17] describes a convolutional neural network (CNN) for sarcasm detection.

D Attention Mechanism

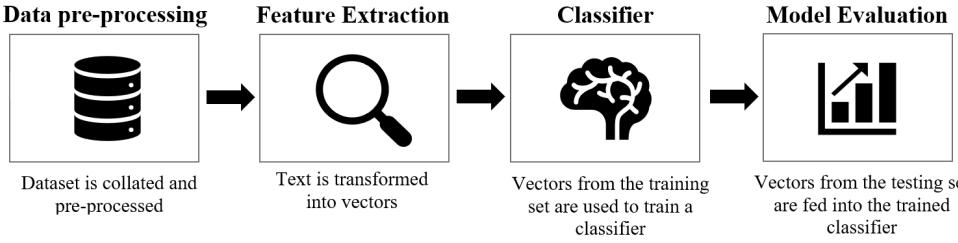
One particularly interesting technique is the *Hierarchical Attention Network (HAN)* defined in Yang et al. (2016) [24] - an attention-based LSTM. The intuition is that certain words and sentences in a document contribute more to its overall meaning, and this is highly context dependent. They included 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.

SOLUTION

This project addresses a binary classification problem, aiming to determine if an unseen snippet of text is sarcastic or non-sarcastic. Our sarcasm detection tool employs trained classification models which form predictions based on knowledge gained from features encountered during training. Developing highly accurate classification models remains the greatest challenge in this project, therefore it is the central focus of our solution. Model peformance is strongly dependent on the data pre-processing and feature extraction techniques employed, as well as the choice of model architecture and training procedures.

In our solution, high quality datasets are first collected and pre-processed. During feature extraction, the textual training data is transformed into numerical vectors and these features are used to train a classification model. The trained classifier is then carefully evaluated in order to account for potential overfitting. The following figure outlines the model pipeline, and each of these stages are further addressed in the remaining sections.

Figure _: Overview of the solution



A Implementation Tools

This project is majoritively written in **Python**, as it has an abundance of open-source libraries well-suited to machine learning and deep-learning tasks. Additionally, Python is very readable, making it easy for others to interpret, test and reproduce code. We use **pandas** for manipulating the datasets, as it provides functionality for applying functions globally across the dataset. We use **pickle** to store the vectorised datasets in memory. Vectorisation is a costly process, therefore writing the files to memory reduces long-term time demands during the experimentation and testing phases. We also incorporate **SpaCy** when cleaning and tokenising the data.

In terms of *classification*, we implement most of the machine learning architectures using **scikit-learn** - an open-source machine learning library. For constructing deep learning models, we use **keras**, as its modular structure allows for extensive experimentation and simplifies tweaking of model architectures.

B Datasets

We utilise three publically available datasets, and a statistical analysis of their properties is provided in *figure* _. This includes the number of instances in the dataset, the average number of tokens per instance and a benchmark score where the dataset was used in the original paper to train one or more classifiers. This data is collected from multiple domains and online platforms, including Twitter, Amazon, and online news outlets; providing broad coverage of sarcasm usage in both formal and informal media.

1. **News Headlines** - *Misra et al. (2019)* [14] collected thousands of news headlines from two sources - *The Onion* [3] (renowned for posting satiricial news) and *The Huffington Post* [4]. News headlines are written formally, hence there is little noise in this dataset due to the absence of erroneous spelling. These headlines are self-contained, unlike tweets which may be in response to another tweet thereby excluding necessary context. Misra et al. (2019) [14] used a hybrid neural network architecture and achieved an *accuracy* of 0.897.

2. **Twitter Data** - One of three datasets collated in *Ptáček et al. (2014)* [18], we use a balanced dataset of 100000 English tweet ids, whereby the original tweet must be scraped using the Twitter API. Of this original corpus, 17.28% of the original tweets remain available to access on Twitter, therefore all others are omitted from the compiled dataset. This dataset is messy, as sarcastic instances are those that include *#sarcasm*, and non-sarcastic instances are selected arbitrarily from the set of all other tweets. Hence, some of the sarcastic instances may not be sarcastic, and some non-sarcastic instances may be sarcastic. We take a subset of the remaining tweets (7500 from each class) to form a smaller balanced corpus. Ptáček et al. (2014) [18] achieved an F_1 score of 0.947.

3. **Amazon Reviews** - *Filatova (2012)* [3] used crowdsourcing to collect Amazon reviews, human-labelled by 5 annotators. The number of stars given to a review, as well as the title and content are provided. These reviews are typically longer than the average tweet, which is capped at 280 characters. The author's punctuation and spelling is preserved, therefore this data is very messy. This is the smallest of the three datasets. This dataset was not used by its compilers for classification, hence there is no score available for comparison.

Figure -: Statistical breakdown of the datasets used in this project

Dataset	Dataset Size	Composition*	Avg. tokens	Benchmark Score
News Headlines	28619	pos: 47.6% neg: 52.4%	11.2	0.897
Twitter Data	17280	pos: 50.4% neg: 49.6%	19.8	0.947
Amazon Reviews	1254	pos: 34.9% neg: 65.1%	276.4	N/A

**pos* indicates positive instances (sarcastic) and *neg* indicates negative instances (non-sarcastic)

[3] www.theonion.com
[4] www.huffpost.com

Each of these datasets are sufficiently large enough to provide a representative view of their respective data sources, without exceeding our computational resources. However, we augment the *Amazon Reviews* corpus using a synonym replacement strategy to increase the amount of data used in deep learning as this improves training stability and model performance. Adjectives and nouns are chosen at random from each snippet of text and are substituted with a synonym so as not to alter the syntax but not the semantics of the text. This procedure is repeated to produce 8 variations of each snippet; hence, the final size of the dataset is 10032.

Most of the datasets are approximately balanced; despite the fact that sarcasm is used infrequently in real-life. On an unbalanced dataset, a classifier can achieve high accuracy simply by annotating each data instance with the majority class label. As most machine learning algorithms are designed to optimise overall accuracy, it is important to provide them with approximately equal amounts of both sarcastic and non-sarcastic data, to prevent them from overfitting to either class.

C Data Pre-Processing

Datasets of user-generated content are inherently noisy and ill-structured, as users are free to include spelling errors and inconsistent capitalisation. Pre-processing is necessary for removing as much of this noise as possible to reduce sparsity in the feature space; although, we must be careful not to remove useful features.

On Twitter, users can include hyperlinks and references to other users in their tweets; however, this can be a source of noise in the dataset therefore we remove them using regular expressions. In *Ptáček et al. (2014)* [18], hashtags are removed in pre-processing. However, *Liebrecht et al. (2013)* [10] showed that the sentiment contained in hashtags can be used to indicate sarcasm, such as the use of #not to suggest insincerity. In the simplest case, single-word hashtags can be included simply by removing the # symbol, however multi-word hashtags must first be parsed to separate the tokens. As this is beyond the scope of this project, we remove all hashtags with the exception of *#not* (which we substitute with *not*).

Punctuation is removed and text is transformed to lowercase in order to reduce sparsity induced by the optional inclusion of punctuation and variations in letter capitalisation. This ensures that are fewer representations of the same word, as tokens such as *'CAN'T'*, *'can't'*, and *'Cant'* are all mapped to the same single token - 'cant'. However, punctuation and capitalisation can be used for emphasis, which may in turn be indicative of sarcasm. Hence, we re-introduce this meaning in the form of additional lexical features using a procedure outlined in the feature extraction section. Finally, we remove numbers and stopwords, such as *'the'*, *'in'* and *'an'*, as they provide little additional meaning to a sentence despite the number of times they occur.

Figure -: Proportion of tokens represented in the GloVe dictionary

Dataset	**Original data**	**Processed data**
News Headlines	97.71%	97.71%
Twitter Data	89.62%	96.43%
Amazon Reviews	83.17%	98.22%

Figure _ shows the proportion of GloVe-representable tokens before and after processing, where the same processing techniques are applied to each dataset. GloVe is a word-embedding technique, detailed in the feature extraction section, where tokens are encoded as pre-computed numerical vectors. We use the number of tokens represented in the GloVe dictionary (containing 27

billion tokens) as a measure of quality, as uncommon or ill-formed tokens are ommitted from this dictionary. The quality of the Amazon and Twitter data vastly improved after processing, however the quality of the news headlines corpus remained consistent as the original data is written formally and in lowercase.

D Feature Extraction

Extracting meaningful data from large corpora is vital for enabling classifiers to make high-quality predictions. As such, it follows that in the context of natural language processing, feature extraction techniques should encode the underlying semantics of text into the numeric vectors they produce. We produce our dominant features using established vectorisation techniques such as GloVe [15] and ElMo [16]. However, we also experiment with a number of custom features in combination with our main features, generated using the procedures outlined in *section D.2*.

D.1 Main Features

Traditional Approaches – As our least advanced feature extraction techniques, *Bag of Words (BOW)* and *Term Frequency-Inverse document frequency (TF-IDF)* [21] are used as a baseline against which to compare more advanced approaches. In BOW, text is encoded as a single vector containing a count of the number of occurrences of each word. TF-IDF extends the BOW model by providing each word with a score that reflects its importance based upon its frequency within the document. These approaches create sparse inefficient representations, where each vector is the size of the vocabulary. Additionally, vectors produced in this way do not preserve the context within which words are found, although sarcasm is a highly contextual phenomenon.

Word Embedding – Introduced in Mikolov et al. (2013a) [13], *Word2Vec* describes a group of related models that were the first to produce high-quality vector representations of words, known as word embeddings. Word2Vec consists of a shallow (two-layer) neural network that takes a large corpus and produces lower-dimensional vectors than the sparse representations generated by BOW and TF-IDF; however, intrinsic statistical properties of the training corpus, which were key to earlier techniques, are neglected, therefore global patterns may be overlooked. To mitigate this, we use the *GloVe* [15] approach, which generates word embeddings by constructing an explicit word co-occurrence matrix for the entire corpus. Word embeddings appear in a high-dimensional vector space, whereby words that share a similar context are clustered together.

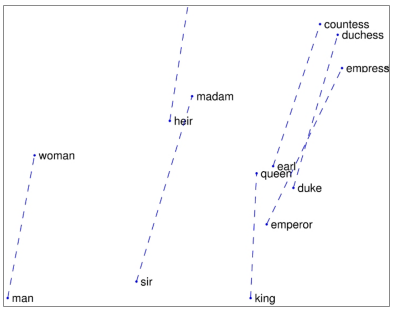


Figure -: GloVe substructure[5]

GloVe 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. $cars - car \approx apples - apple$. GloVe is used in this project to produce word embeddings for each corpus, however it does not accommodate polysemy, which describes the co-existence of alternative meanings for the same word, and cannot generalise to words that were not specifically included in the training set.

[5]https://nlp.stanford.edu/projects/glove/

Contextualised Word Embedding – Peters et al (2018) [16] introduced the *ELMo* model, and showed that ELMo markedly improved the state-of-the-art in a number of NLP tasks.

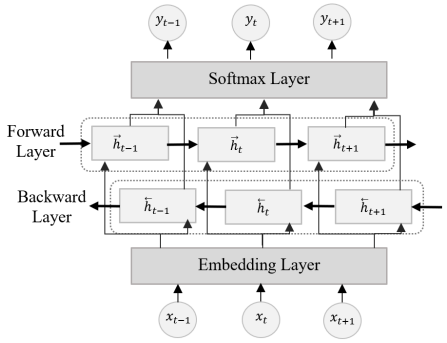


Figure -: ElMo model pipeline

ELMo utilises 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. 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. These state-of-the-art contextualised embeddings are used in this project as the most advanced method of feature extraction.

D.2 Miscellaneous Features

While deep learning algorithms are able to extract useful higher-level features from dense vectors (such as ElMo and GloVe), machine learning classifiers may require additional handcrafted features to better inform their predictions. We consider miscellaneous features as well as those produced by the aforementioned techniques - including lexical (punctuation) features, topic features and sentiment features. We produce low-dimensional feature vectors for each of our datasets, concatenating those that are shown to improve classifcation scores with existing vectors.

1. Punctuation features – We extract punctuation features following the procedure outlined in Tsur et al. (2010) [23]. Five lexical features are combined to produce a 1×5 dimensional vector for each text snippet, including the length of the snippet in tokens, as well as the number of: "!" characters, "?" characters, quotations and tokens that include at least one capitalised character. These features are normalised by dividing them by the (maximal observed value $\cdot$ average maximal value of remaining feature groups).

2. Topic features – We use Latent Dirichlet Allocation (LDA) to perform topic modelling - this is an unsupervised machine learning technique where a fixed number of abstract "topics" are identified in a corpus. Each instance in a dataset is represented as the distribution of its topics, relative to the topics in the global dataset. The intuition is that positive and negative instances may each contain their own distinct assortment of topics, thereby informing future predictions on unseen sarcastic and non-sarcastic data.

3. Sentiment features – We produce sentiment features using the state-of-the-art *SentimentAnnotator* [22] by Stanford CoreNLP [11] which assigns a discrete sentiment label to text, where 0 is very negative and 4 is very positive.

Example: "i love when my train is late"

Tokens	**i**	**love**	**when**	**my**	**train**	**is**	**late**
Sentiment	2	4	2	2	2	2	1

Sentiment	**0**	**1**	**2**	**3**	**4**
Frequency	0	1	6	0	1
Proportion	0	0.125	0.750	0	0.125

Overall sentiment: 3.0
Feature vector: [0.0, 0.125, 0.750, 0.0, 0.125, 3.0]

For each instance in the dataset, we collect the sentiment values for each of the individual tokens, as well as the overall sentiment of the snippet. We contruct 1×6 dimensional feature vectors, where values 0 - 4 show the distribution of sentiment classes for tokens in the snippet, and value 5 shows the overall sentiment label.

E Binary Classification

In this section, we describe the machine learning and deep learning classifiers used in this project. These algorithms are trained on a set of rich features. For example, our machine learning classifiers are trained on Bag of Words, TF-IDF, GloVe and ElMo vectors, and we experiment with concatenating miscellaneous features where they have been proven to be informative in our binary classification task. We train each deep learning model on GloVe and ElMo vectors only, as it is computationally intractable to train neural networks on high-dimensional sparse features.

E.1 Machine Learning Models

We experiment with five candidate machine learning classifiers, evaluating each of them when trained on four types of features. Each of these supervised models performs binary classification, where unseen samples are predicted to be sarcastic or non-sarcastic. Brief descriptions of the models are provided below in the context of our specific binary classification problem.

1. **Naïve Bayes Classifier –** A group of probabilistic models that use Bayes theorem to make predictions; variations include Multinomial, Bernoulli and Gaussian Naïve Bayes. The naïve assumption that all features are independent of one another allows an estimate of $P(Sarcastic \mid x)$ (i.e. probability that a statement is sarcastic, given its feature vector, x) to be made using Bayes theorem.

2. **K-Nearest Neighbours (KNN) –** A non-parametric model which uses lazy learning at the time of prediction. An unseen sample is assigned the majority class label amongst its k nearest neighbours, where k is a hyperparameter to be finetuned. We found that k=5 achieved the optimal balance between the computational cost of a large number of neighbours and increased noise and bias associated with small values of k.

3. **Support Vector Machine (SVM) –** Uses a kernel function to draw a hyperplane which divides a feature space into two subspaces. Features at the shortest euclidean distance (*margin*) to the hyperplane are known as support vectors, and their distance from the hyperplane is maximised to create a decisive boundary between the sarcastic and non-sarcastic data. Our SVM uses a linear kernel as this model performed better than the SVM with radial basis function (RBF) kernel.

4. **Logistic Regression –** Models a binary dependent variable by finding a relationship between features and the likelihood that they belong to either class. It is a more advanced approach than linear regression, which aims to model data using a linear equation.

5. **Random Forest Classifier –** An ensemble learning method which fits a number of decision trees on subsamples of the training set. The decision trees produce class predictions which are condensed into a single prediction by means of voting. As they are largely independent of one another, this low correlation allows them to produce collective predictions that outperform their individual predictions.

E.2 Deep Learning Models

Deep learning classifiers do not require the hand-crafted features used in supervised classifiers, and are instead capable of learning higher-level features from dense vectors such as ElMo and GloVe. We focus on two categories of neural networks - Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). While it is beyond the scope of this paper to describe in detail the concepts involved in CNNs and RNNs, we provide a brief explanation below. As we are performing binary classification, we use the Binary Cross-Entropy loss function.

Convolutional Neural Networks (CNN) [9] were popularised for their applications in computer vision and were designed to be used with image data [8]; however, they can be adapted for text-based classification tasks by replicating the two-dimensional structure of images (*figure _*).

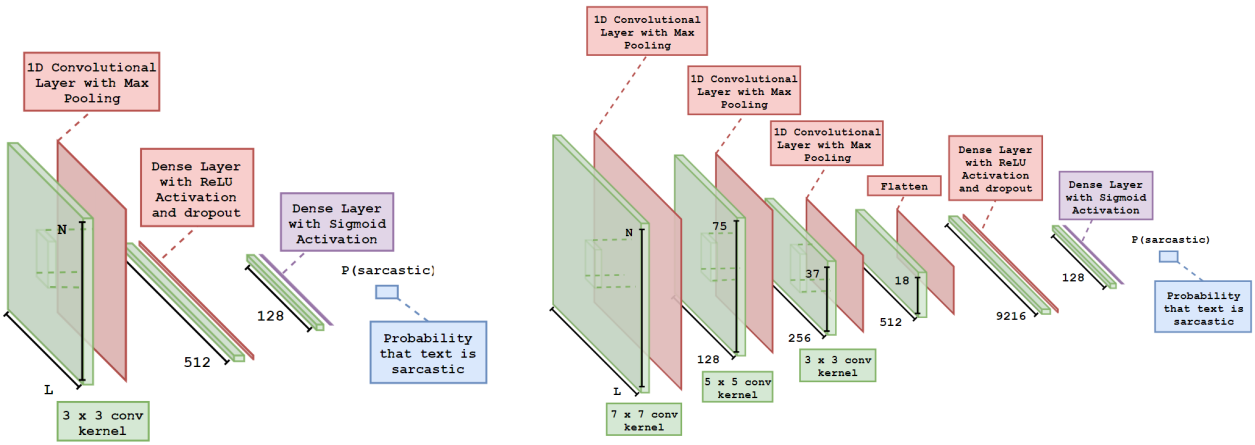
$$\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,N} \\ \vdots & \vdots & \vdots & \vdots \\ w_{M,1} & w_{M,2} & \dots & w_{M,N} \end{bmatrix}$$

Figure -: $m \times n$ feature vector

To create input features for the CNN, each token in a sentence is represented independently by a fixed-size n-dimensional vector; as such, the sentence is represented by an $m \times n$ vector of feature weights, where m is the number of tokens in the sentence.

The characterising feature of a CNN is the convolutional layer, where filters are applied to feature vectors using the mathematical operation of convolution, and a max pooling operation is applied over each resulting feature map. Tokens in a sentence occur on a time axis, as the order that the tokens appear in gives rise to their meaning; hence, the filter slides downwards over the rows of the vector (*maximum pooling over time*) to produce a single output value. This is especially useful for classification tasks, where the desired output is a fixed-sized vector representing a class. Due to the high speed of the convolution operation, many convolutional layers can be used in a single network without excessivly slowing the training process.

We implement two convolutional neural networks and present their architectures in *figure _* and *figure _*. We refer to one as a simple CNN as it contains only one convolutional layer, and to the other as a deep CNN as we incorporate three convolutional layers. With over 1.7 million trainable parameters, our deep CNN is more computationally complex and is slower to train than our simple CNN, which contains approximately 140 thousand trainable parameters. Both networks take an $N \times L$-dimensonal vector, where N is the length of each token vector, and L is the number of tokens in the snippet.



(a) Simple CNN Architecture

(b) Deep CNN Architecture

Figure -: Illustration of Convolutional Neural Network Architectures

Recurrent Neural Networks (RNN)

outputs from previous time steps are re-introduced into the network

contain an additional hidden state that is updated as inputs are fed through the network.

RNNs use feedback loops to process sequential data gradient-based learning, where This creates a form of memory, as it allows information to persist in the network.

RNNs use backpropagation to calculate the gradient of the loss function with respect to individual network weights, and these weights are updated in a manner that is proportional to the gradient. However, RNNs are susceptible to the *vanishing gradient problem*, more so when we continue to add complexity (in the form of additional layers) to our network. This is where the gradient with respect to the earliest layers becomes negligible; as do the updates made to their weights. The learning process may become unstable and the network will have a short-term memory.

LSTMs and GRUs have internal mechanisms known as gates, and can overcome the pitfall of short-term memory.

A directed cycle is formed within an RNN whereby former outputs are input back into a hidden unit.

This hidden state represents the context based on prior inputs, giving rise to the *temporal dimension* in RNNs which allows them to take into account time and sequencing. This leads to the interesting property that the same input could produce a different output depending on the previous inputs given to the RNN.

In a vanilla RNN, the input and hidden state are simply propagated through a single tanh layer. However in a Long Short Term Memory model (LSTM), three gates are introduced, as well as a cell state, allowing an LSTM to better preserve long-term dependencies. We implement a Vanilla RNN as a baseline against which to compare our more advanced RNN models, as well as three additional models as described below:

1. **Gated Recurrent Unit –**
2. **Long Short-term Memory *(LSTM)* –** three gates are introduced, as well as a cell state, allowing an LSTM to better preserve long-term dependencies

3. **Bidirectional Long Short-term Memory *(BiLSTM)*** – Uses two LSTMs and concatenates their output sequences, where one is trained on reversed input sequences and the other is trained on the original input sequences. This allows context to be captured from both directions, and can lead to improved predictions.

RNNs have been successfully applied to language-related tasks. Often, we may require more context than just the most recent surrounding text. The gap between the most relevant information needed to form a prediction can be very wide, if the relevent contextual information was given at the beginning of the paragraph. LSTMs are especially suited to 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 [7] instead which are more effective at preserving long term dependencies.

F Attention

When humans read text to answer a query or to inform a decision, we automatically focus more on specific regions, such as the text surrounding key words or phrases. This is known as *human visual attention* and has been well studied by psychologists; however, we can incorporate a similar attention mechanism into our recurrent deep learning models. This provides insight into which tokens contribute more or less in classification decisions. Additionally, incorporating an attention mechnism can lead to better classification performance. Extending our binary classification solution in this way enables us to answer the research question: *Can a custom model identify which linguistic cues correlate more to sarcastic labels?*

We employ the attention mechnism outlined in Yang et al. (2016) [24] which uses context to identify relevant tokens, rather than filtering for key words that are taken out of context. The meaning of words depends on the context in which they are found; hence, harnessing as much context as possible is key to informing classification decisions and producing meaningful attention visualisations. We use an ElMo embedding layer to produce contextualised word embeddings from raw text, as they encode the context in which a word is found (polysemy) and can generalise to out-of-vocabulary tokens. We feed these word embeddings through a bidirectional gated recurrent unit in order to retain both the left and right contexts of each word.

Output sequences from the bidirectional GRU are passed through our attention layer, which returns a context vector and attention weights. We can visualise our attention weights by projecting them into a colourspace and displaying the tokens highlighted in their respective colours, where darker colours indicate a higher-level of attention.

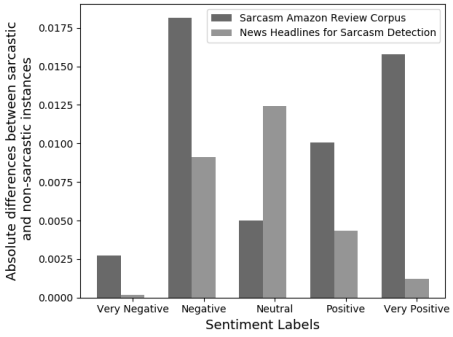


Figure -: Visualisation of attention weights

This paper considers that documents have a hierarchical structure

The intuition behind this is such that not all components of text are equally relevant for indicating sarcasm or not sarcasm.

We implement the attention mechanism described in Yang et al. (2016) [24] in the form of an Attention layer. This is an intermediate layer placed after the bidirectional LSTM layer in our model.

In Yang et al. 2016, they used two sentence-level and word-level attention

Visualising the output of our attention layer shows that the model places more emphasis on qualitatively informative tokens.

The attention mechnism has played a fundamental role in recent deep learning research, fuelling powerful language models such as BERT.

LSTMs capture long-term dependcies, more so than a RNN. An Attention mechanism attributes more importance to a subset of the input words.

Bahdanau et al. (2015) suggested that a context vector takes into account all of the input words, assigning relative importance to some more than others.

We take the dot produce of weights and inputs and add our bias terms. We then apply a tanh activation and softmax gives the alignment scores.

We take the dot product of the alignment scores and the hidden states to produce a context vector which is propagated through the remainder of the network.

This attention mechanism uses all previous hidden states of the LSTM

RESULTS

A Evaluation Method

This section discusses our approach to evaluating the performance of the trained models. Each trained model makes predictions on labelled unseen data, and these predictions are compared to the corresponding ground truth labels. From this, we extract the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). We use four metrics that leverage this data differently to produce an evaluation of the trained models.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$
$$F_1\ Score = \frac{2 \times precision \times recall}{precision + recall} \qquad MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Precision refers to the proportion of classified-sarcastic data that is *truly sarcastic* (how many positives are true positives), and recall describes the proportion of the truly sarcastic data that is classified as such (how many true positives are labelled as positives). We also use the harmonic mean of precision and recall (F_1 score), however this metric has faced some criticism for giving equal weight to precision and recall [6], therefore we also consider these measures separately. The Matthews Correlation Co-efficient (MCC) is a balanced measure which shows high scores if the true positive and negative rates are high and if the false positive and negative rates are low.

B Analysis of Miscellaneous Features

We analyse the effectiveness of sentiment, punctuation and topic modelling features on each of our training datasets by training logistic regression models using these features in isolation - N.B. logistic regression is used as these feature vectors are small. We use 5-fold cross-validation to compute the F_1 scores of these models. A classifier that predicts classes randomly can achieve an F_1 score of 0.5 on a balanced dataset - hence, we consider models achieving greater than 0.5 to be indicative of useful training features.

Table -: F_1 score of logistic regression models trained on hand-crafted features

Feature	News Headlines	Twitter Data
Sentiment	0.516	0.593
Punctuation	0.344	0.611
Topic	0.525	0.530

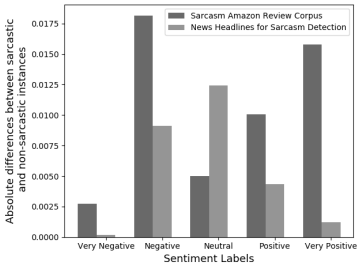


Figure -: Absolute differences

All three categories of features were more informative when extracted from Twitter data, and this may be due to the informal and irregular nature of user-generated tweets. Sarcasm in tweets is more overt than in News headlines, which are typically contextual and subtley sarcastic. This enables us to assess whether these features alone are indicative of sarcasm in different media types.

For the Sarcasm Amazon Review Corpus, within most sentiment classes there is a greater difference in the proportion of tokens for non-sarcastic and sarcastic instances. This indicates that sarcastic and non-sarcastic instances are easier to distinguish, given that their sentiment compositions are more distinct. The News Headlines for Sarcasm Detection results displays the opposite, and this explains why the logistic regression classifier trained on this data achieved an F_1 score of only 0.515, as the features for each class are very similar.

C Binary Classification

To compare the performance of our machine learning and deep learning classifiers, each model is trained independently on numerous features extracted from our datasets. For each dataset, we report on only a subset of our most informative results, including the models with the highest F_1 scores. We contrast the performance of our machine learning classifiers with our deep learning classifiers in order to answer the research question: *Do deep learning techniques perform better than machine learning approaches?*

Experimental Settings – For our machine learning experimentation, 5-fold cross validation is employed to ensure reproducability and consistency of results. However, it is infeasible to apply the same approach to our deep learning experimentation, as training is substantially slower. We train each deep learning model on GloVe and ElMo vectors only, as it is computationally intractable to train neural networks on high-dimensional sparse features. The upper-bound on the number of training epochs is 300, although this is seldom reached due to the use of the *Early Stopping* callback function.

1. **Early stopping –** Training is interrupted when the validation loss stops declining - this is used to determine the appropriate number of epochs.
2. **Model Checkpointing –** We track and save the weights of the best performing model during training i.e. the model with the lowest validation loss.

Model performance peaks early in the training process for all of our models, although there is some fluctuation in the optimal number of epochs for different model types, hence early stopping is incorporated to stop training once model performance begins to plateau. The number of epochs for which no improvement in validation loss is seen (*patience*) is set to 10. Setting the patience too low can hamper a model that plateaus for a while before improving again. We allocate 90% of data to the training set, and the remaining 10% to the testing set.

C.1 News Headlines Dataset

Table -: Classification results on the *News Headlines* dataset

Machine Learning Model	**Feature(s)***	**Precision**	**Recall**	**F_1 Score**
Random Forest Classifier	GloVe	0.774	0.735	0.754
Support Vector Machine	TF-IDF	0.688	0.751	0.718
Random Forest Classifier	ElMo	0.801	0.793	0.797
Logistic Regression	ElMo	0.840	0.852	0.846
Logistic Regression	ElMo · S · T	**0.848**	**0.854**	**0.851**
Deep Learning Model	**Feature(s)**	**Precision**	**Recall**	**F_1 Score**
CNN	GloVe	0.818	0.806	0.812
LSTM	ElMo	0.843	**0.878**	0.860
Bidirectional LSTM	GloVe	0.855	0.873	0.864
Deep CNN	ElMo	0.873	0.869	0.871
Bidirectional GRU	ElMo	**0.889**	0.877	**0.883**

**S* refers to sentiment features and *T* refers to topic features

Deep learning models routinely outperform Machine learning models. Logistic regression trained on ElMo with sentiment and topic features was the machine learning model with the highest F_1 score, and a slight improvement of 0.1%. The Random Forest Classifier was the next best model architecture on this dataset, although its F_1 score was 5.9% lower than the best performing model.

Our advanced RNN models exceeded the F_1 score of the vanilla RNN by an average of _%, which achieved an F_1 score of _. Similarly, our deep CNN achieved an 8.2% increase in F_1 score compared to our simple CNN, as well as more consistent values for precision and recall.

C.2 Twitter Dataset

Table -: Classification results on the *Twitter* dataset

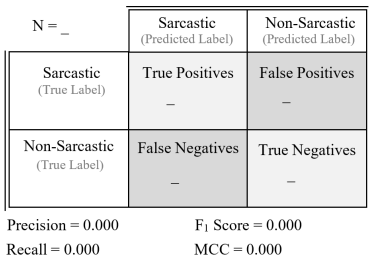
Machine Learning Model	**Feature(s)***	**Precision**	**Recall**	**F_1 Score**
Gaussian Naïve Bayes	Bag of Words	0.677	0.685	0.681
Random Forest Classifier	GloVe	0.669	0.623	0.645
Support Vector Machine	ElMo	0.722	**0.789**	0.754
Logistic Regression	ElMo	0.758	0.766	0.762
Logistic Regression	ElMo · S · T · P	**0.794**	0.786	**0.790**
Deep Learning Model	**Feature(s)**	**Precision**	**Recall**	**F_1 Score**
Vanilla RNN	ElMo	0.834	0.801	0.817
Bidirectional LSTM	GloVe	0.829	0.815	0.822
CNN	ElMo	**0.851**	0.814	0.832
Bidirectional LSTM	ElMo	0.847	**0.851**	**0.849**
Long Short-Term Memory	GloVe	0.837	0.845	0.841

**S* refers to sentiment features, *T* refers to topic features and *P* refers to punctuation features

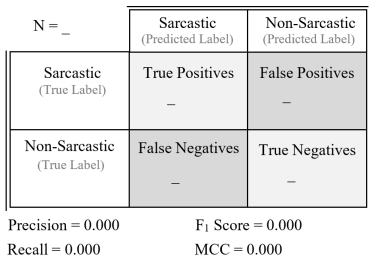
Interestingly, the Random Forest Classifier was the best performing model using GloVe vectors. GloVe performance was weak on this particular dataset, and this could be due to the fact that this dataset has the lowest percentage of tokens present in the GloVe dictionary; hence, we lose the meaning of these tokens in the embedding input.

C.3 Cross-Dataset Evaluation

Given that the following model performed best, trained separately on all three datasets. We evaluate which of these models had the best performance on the other two datasets - this gives us an indication of which model is more likely to generalise to unseen data in multiple forms of media (i.e. which model is most adaptable, therefore the most suitable to be used in our tool)



(a) Trained on News Headlines



(b) Trained on Twitter Data

Figure -: Confusion Matrices of models evaluated on Amazon Reviews

D Attention Mechanism

The superior model in the binary classification task is shown to be the Bidirectional Gated Recurrent Unit (BiGRU) model, trained on ElMo vectors from the News Headlines dataset. Hence, we incorporate our attention mechanism into this model to identify the tokens that lead to positive and negative classification decisions. We also examine whether incorporating attention can boost model performance as hypothesised.

Table -: Comparison of classification results with and without Attention

Model	**Precision**	**Recall**	**F_1 Score**
Bidirectional GRU	0.889	0.877	0.883
Bidirectional GRU with Attention	0.889	0.877	0.883

EVALUATION

In the following section, we analyse the effectiveness of the solution, discussing its strengths and weaknesses and evaluating to what extent it satisfies the research questions and deliverables.

A Solution Strengths

We found that sentiment features can be useful predictors of sarcasm in *some* forms of media. This may be due to the fact that sarcasm must be more overt in product reviews in order to mock the product, rather than the subtle, more contextualised instances in News Headlines.

B Solution Limitations

It was only feasible to evaluate a small range of online media types, however surveying a broader range of media types could give us a wider perspective of sarcasm in online media.

C Lessons learnt

We learned that

D Project organisation and approach

CONCLUSIONS

In terms of satisfying the objectives, this project was an overall success. With an interactive console, it may be useful to individuals who struggle to detect sarcasm in their daily-life, or for the industrial applications of enhanced opinion mining.

The main contributions of this project are as follows: sentiment features are combined with state-of-the-art vectorisation approaches.

For future work, it would be interesting to explore the use of sarcasm in a wider variety of media types, and to extend support to additional non-English languages. Exploring the use of transfer learning with Transformers (BERT) may also produce interesting results.

Table 1: SUMMARY OF PAGE LENGTHS FOR SECTIONS

	Section	Number of Pages
I.	Introduction	2–3
II.	Related Work	2–3
III.	Solution	4–7
IV.	Results	2–3
V.	Evaluation	1-2
VI.	Conclusions	1

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] Elena Filatova. Irony and sarcasm: Corpus generation and analysis using crowdsourcing. In *Lrec*, pages 392–398. Citeseer, 2012.

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

[5] 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.

[6] 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.

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

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

[11] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[12] 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.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[22] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[23] 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.

[24] 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.

[25] Meishan Zhang, Yue Zhang, and Guohong Fu. Tweet sarcasm detection using deep neural network. In *Proceedings of COLING 2016, The 26th International Conference on Computational Linguistics: Technical Papers*, pages 2449–2460, 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.