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 is a unique form of language that cannot be interpreted at surface-level. 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 – Despite the challenges in this domain, this ultimate aim of this project is to produce a tool that can detect sarcasm with a *high degree* of accuracy.

Method – State-of-the-art contextualised word embeddings and deep learning classification models are used to realise this aim.

Results – Through extensive experimentation, we show that

Conclusions – In summary, we conclude that

Keywords — Machine learning, Deep learning, Sarcasm Detection, Sentiment Analysis, 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 sentiment analysis task.

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 sarcastic intent; due in part to the lack of an all-encompassing, universal definition of sarcasm. 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.

B Research Questions and Objectives

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

- *Do deep learning techniques perform better than machine learning approaches?*
- *Can we improve the solution by incorporating sentiment features?*
- *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 experiment with and evaluate machine learning architectures on these datasets. This is achieved by ...

The **intermediate** objectives of this project are to experiment with and evaluate deep learning 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 deep neural network in order to identify words that correlate more to sarcastic labels, in order to produce a visualisation of attention words. This is achieved by...

RELATED WORK

In this section, we compare classification model architectures used specifically in sarcasm detection research, as well as techniques that have seen success in other natural language processing (NLP) classification tasks. In most approaches, 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 often used interchangeably [22]. For clarity, an overview of their definitions are given in figure 1. Both definitions capture the humourous intent in both sarcasm and irony, however sarcasm extends this definition to include the potential for criticism - often present in opinion-based content such as product reviews.

Figure 1: Comparison of defintions of Irony and Sarcasm

Term	Definition
Irony	The use of words that are the opposite of what you mean, as a way of being funny. [1]
Sarcasm	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.[2]

A Traditional Classifiers

A number of simple linguistic approaches have been used in previous research. One such class of naïve approaches is *rule-based*, where text is classified based on a set of linguistic rules. Maynard and Greenwood (2014) [11] 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, whereby if the sentiment of a tweet contradicts that of the hashtag, then this tweet is labelled sarcastic. They reported an F_1 score of 91.03% on a random sample consisting of 400 tweets.

In Riloff et al. (2013) [19], 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 sentiment phrase. Rule-based techniques are fairly primitive when compared to their modern, deep learning counterparts, as each ruleset must be generated manually and for each dataset.

B Machine-Learning Classifiers

A machine-learning algorithm can learn patterns that are associated with sarcastic and non-sarcastic 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, or an effective

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

unsupervised-learning algorithm is used, then the model can begin to make accurate predictions. There are a number of commonly used architectures of machine-learning classifiers, including Naïve Bayes, Support Vector machines (SVMs) and logistic regression based classifiers. Although, not all approaches fit neatly into these categories.

Tsur et al. (2010) [22] 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 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 neighbor (kNN), whereby a sarcasm score is assigned to an unseen vector based upon the weighted average of the k-nearest neighbors in the training set. They reported an F_1 score of 82.7% on the binary classification task

Naïve Bayes classifiers are supervised algorithms that use bayes theorem to make predictions. Reyes et al. (2013) [18] 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.

Support Vector Machines (SVM), can be effective when smaller amounts of training data are available, however they require more computational resources than naïve bayes. SVMs use kernel functions to draw hyperplanes dividing a space into subspaces. González-Ibánez et al(2011) [5] used two classifiers, support vector machine with sequential minimal optimisation, 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 optimisation and unigrams. *Ptáček et al. (2014)* [17] also used support vector machine 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 [16, 25]. 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.

One particularly interesting technique is the *Hierarchical Attention Network (HAN)* defined in Yang et al. (2016) [23] - 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)* [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

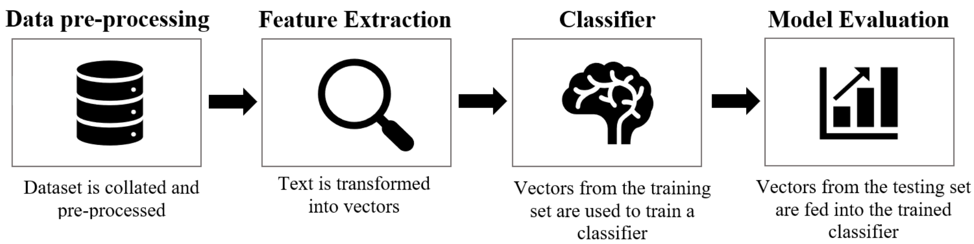
RNN used to overcome the vanishing gradient problem. *Zhang et al. (2016)* [24] used a bidirectional gated RNN to capture local information in tweets, as well as a pooling neural network to extract context from historic tweets, achieving 78.55% accuracy.

Convolutional Neural Networks allow us to extract higher-level features, and they are based on the mathematical operation of convolution. *Zhang et al. (2015)* [25] 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. CNNs have been previously used in sarcasm detection. For example, *Poria et al. (2017)* [16] describes a convolutional neural network (CNN) for sarcasm detection.

SOLUTION

This project addresses a binary classification problem, aiming to identify if a given snippet of text is sarcastic or non-sarcastic. First, the data is collected and pre-processed. Then, features are extracted and the data is vectorised. These vectors are used to train to a classifier, which 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 2: Overview of the model pipeline



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. The model interface is written using web-application development languages such as HTML, CSS and JavaScript.

B Datasets

In similar studies, large datasets have been collected, and in some cases, made publically available. We evaluate the suitability of many of these datasets and select four which are used in this project. We experimente with data from multiple domains and online platforms, including Twitter, Amazon, Reddit and News outlets. This gives a broad coverage of the way sarcasm is used under different circumstances and in both formal and informal media. We evaluate many datasets and their suitability based on statistics from the dataset and desired attributes. For example, we avoid using datasets containing tweet ids, whereby the original tweet must be scraped using the Twitter API. A small portion of the original tweets are no longer available on twitter, therefore this does not make for a fair comparison between our results and the results of the source study.

Four datasets are selected, and a statistical break down of their properties is provided in figure 3, including the number of instances in the dataset, the average number of tokens in each instance and the percentage of tokens that are included in the dictionary. GloVe is a method of feature extraction where word embeddings for tokens are provided as a dictionary. The number of tokens present in the GloVe dictionary is a measure of quality, as this method of feature extraction cannot be used on instances that contain many ill-formed and unrepresented tokens. Additionally, all tokens are converted to lowercase beforehand to make for a fair comparison between datasets, as only lowercase tokens are present in the GloVe dictionary. A detailed explanation of GloVe is provided in the feature extraction section.

Figure 3: Statistical breakdown of the datasets used in this project
N.B. *pos* indicates positive instances and *neg* indicates negative instances

Dataset Title	**Number of data points**	**Avg. #tokens**	**% in GloVe dictionary**	**Benchmark**
News Headlines Dataset For Sarcasm Detection	28619 pos: 47.6% neg: 52.4%	11.2 tokens	97.8%	89.7%
Sarcasm Amazon Review Corpus	1254 pos: 34.9% neg: 65.1%	276.4 tokens	98.2%	N/A

1. **News Headlines Dataset For Sarcasm Detection**
   This dataset was collated for use in *Misra et al. (2019)* [13]. Headlines are collected from two news sources - *The Onion* [1] and *The Huffington Post* [2]. The Onion is renowned for posting satiricial news, and these sarcastic headlines account for the sarcastic instances in the dataset. The non-sarcastic instances were collected from the Huffington Post. There is little noise in this dataset, as news headlines are written formally, and are far less likely to contain spelling and grammar errors than data from more informal platforms. These

[1] www.theonion.com
[2] www.huffpost.com

headlines are self contained, unlike tweets which may be in response to another tweet, thereby excluding necessary context. Misra et al. (2019) [13] used a hybrid neural network architecture and achieved an *accuracy* of 89.7%. NB. Accuracy was used as the dataset is approximately balanced.

2. **Sarcasm Amazon Review Corpus**
This corpus was collected using the procedure outlined in *Filatova (2012)* [3]. Amazon reviews are collected via crowdsourcing, and are human-labelled by 5 annotators. The number of stars given to a review, as well as the title and content are provided. The author's punctuation and spelling is preserved, therefore this data is very messy.

C Data pre-processing

Data pre-processing is an important stage in natural language processing, with the potential to hinder or boost model performance. Datasets in this domain are inherently messy, often because they are collated from user generated content. Hence, data pre-processing is vital to reduce noise and sparsity in the feature space caused by inconsistent letter capitalisation, varying use of punctuation and erroneous spelling. The suitability of each technique is highly dependent on the dataset and on the feature-extraction technique, as certain types of pre-processing may result in the removal of useful features.
Reducing sparsity in the dataset is vital in order to use word embeddings such as Word2Vec and GloVe. They cannot generalise to out-of-vocabulary tokens i.e. tokens that were not in the original training set, and in the GloVe dictionary, all tokens are given in lowercase. As a result, it is essential that the data is converted to lowercase, however capitalisation can indicate emphasis, which may be indicative of sarcasm. N.B. It is necessary to re-introduce this meaning in some form. We add the token ! before a capitalised word, indicating emphasis, or "shouting". 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)* [17] 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)* [10] 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 capatilisation 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 capitalisation can be used for emphasis, therefore removing these attributes may make the presence of sarcasm less obvious.

In the *News Headlines Dataset For Sarcasm Detection* dataset, the raw data was pre-cleaned, as it had all been transformed into lowercase. As news headlines are written in a formal manner, spelling and grammar errors are rare, thereby increasing the likelihood that pre-trained embeddings are present in the GloVe dictionary.

D Feature Extraction

Extracting meaningful data from large corpora is a complex task, however in order to do this successfully, we first construct word vectors that capture semantic and syntactic relationships. The textual data is transformed into numerical vectors, so as to encapsulate the seminal features of language.

Traditional Approaches – We utilise a number of traditional vectorisation methods - perhaps the simplest of which is *Bag of Words (BOW)*, whereby a snippet of text is encoded as a single vector containing a count of the number of occurrences of each word. *Term Frequency-Inverse document frequency (TF-IDF)* [20] extends the BOW model by providing each word with a score that reflects its level of importance based upon its frequency within the document. These approaches have two main disadvantages, firstly, each vector is the size of the vocabulary, creating sparse inefficient representations. Secondly, vectors produced in this way do not preserve the context within which words are found, and sarcasm is a highly contextual phenomenon.
The *Bag of N-grams* approach is a generalisation of the BOW approach, 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 can result in an even larger and sparser feature set.

Word Embedding – Embeddings produce lower-dimensional vectors than the sparse representations generated by earlier techniques. Introduced in Mikolov et al. (2013a) [12], *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, such that words that share a similar context are clustered together in the feature space.

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* [14] 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). An example is given in figure _. This allows it to capture both global and local statistics of the corpus. GloVe is used in this project to produce word embeddings for each corpus, as some of the datasets have a high average number of tokens in each snippet, therefore local context is insufficient.
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.

Figure -: Word co-occurrence matrix with window size 1 for the snippet:
"the clouds are in the sky"

	the	clouds	are	in	sky
the	0	1	0	1	1
clouds	1	0	1	0	0
are	0	1	0	1	0
in	1	0	1	0	0
sky	1	0	0	0	0

Given a corpus containing W tokens, a $W \times W$ matrix is constructed. The value in cell (x,y) denotes the frequency with which token *x* has co-occurred with token *y* within a fixed-size window. From this co-occurrence matrix, the probability of token *x* co-occuring with token *y* is given by the formula:

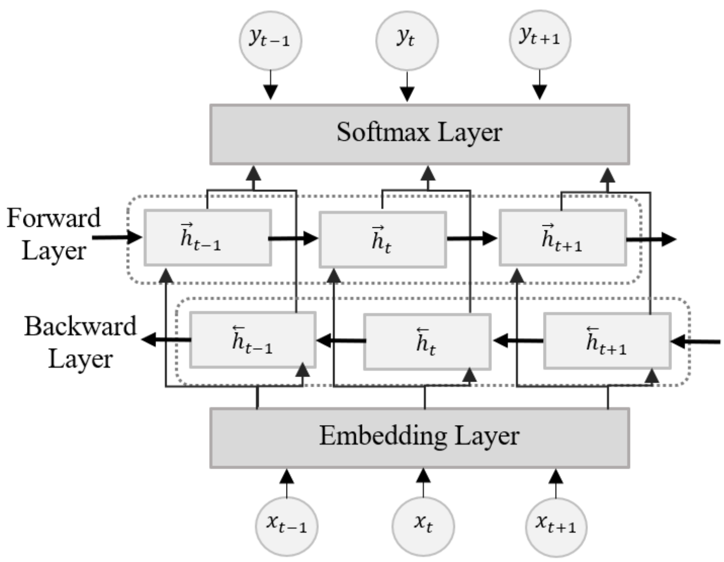
$$P_{co}\left(t_x|t_y\right) = \frac{C(t_y,t_x)}{C(t_y)}$$

where $C(t_i, t_j)$ is the number of times token *i* co-occurs with token *j*.

The word meanings are encapsulated by the ratios of co-occurence probabilties, rather than the probabilities themselves.

Contextualized Word Embedding – In pursuit of a more robust approach, we look to deep neural language models. Peters et al (2018) [15] 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 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.

Figure -: Overview of the ElMo model pipeline



Miscellaneous Features – We consider a number of linguistic features in combination with the vectors produced by the aforementioned vectorisation approaches, by concatenating both vectors. For example, meta-data can be used as features for datasets containing additional information, e.g. Sarcasm Amazon Review Corpus, using the number of stars in a review as an additional feature. As these features are manually selected for the domain, and are closely related to the type of data contained in the dataset, these features are outlined in the results section alongside the experiments within which they were used.

E Classification

In this section, machine-learning and deep-learning classification models are considered. These algorithms are trained a set of rich features as outlined in the previous section.

E.1 Machine Learning Models

In terms of machine learning models, we experiment with supervised and unsupervised learning (k-means clustering)

1. **K-Means Clustering –** As an unsupervised algorithm, K-Means clustering does not require labelled training data. K random cluster centres, or centroids, are selected. Then, the set of observations (training data) is partitioned into k clusters, whereby each data point belongs to the nearest centroid. This will make for an interesting comparison to the remaining models, which rely upon labelled training data.

2. **Support Vector Machine –** Support vector machines are a popular model for binary classification tasks, where a hyperplane is drawn such that the data points are separated into two groups. For the specific task of sarcasm detection, these categories are sarcastic and non-sarcastic. The distance from data points to the hyperplane is known as the margin, and the hyperplane is drawn such that the margin between classification groups is maximised.

3. **Naïve Bayes Classifier –** This is a probabilistic model derived from Bayes theorem, where all features are assumed to be independent of one another. Commonly used in the document classification problem, we calculate $P(Sarcastic|features)$. This is the probability of an outcome A (i.e. statement is sarcastic), given its feature vector, x. There are a number of variations of Naïve Bayes - including Multinomial, Bernoulli and Gaussian.

4. **Logistic Regression –** A statistical model that aims to predict the likelihood of an event occuring given some previous data. It works with binary data, i.e. is the statement sarcastic or not. It is more advanced than linear regression, which aims to model data using a linear equation.

5. **Random Forest Classifier –** An ensemble learning method, the random forest classifier fits a number of decision tree classifiers on subsamples of the training set. The decision trees are fairly independent of one another, and this low correlation allows them to produce ensemble predictions that outperform their individual predictions. Where one model goes wrong, the others are able to contradict this incorrect prediction.

E.2 Deep Learning Models

Deep learning classifiers do not need to use hand-crafted features used in supervised classifiers.

E.2.1 Recurrent Neural Networks – In a Recurrent Neural Network (RNN), prior inputs are used to inform future outputs. As in all neural networks, input vectors are modified by weighted nodes as they travel through the network. In a RNN, there is an additional hidden state which represents the context based on prior inputs, and is updated by inputs as they are fed through the network. This gives way 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.

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.

E.2.2 Convolutional Neural Networks – A Convolutional Neural Network (CNN) [9] is a specialised form of neural network. They were popularised for their applications in computer vision, hence they are designed to be used with two-dimensional image data [8], but have since been adapted for text-based classification tasks. CNNs consist of a few layers of convolutions with non-linear activation functions - they have fewer layers than typical neural networks, where each layer applies dfferent filters and combines their result. The convolution operation is very fast, therefore many convolutional layers can be used in a single network.

Figure -: $m \times n$ feature vector

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

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. This feature vector is used as input to the convolutional neural network.

Convolutional layers compute the dot product of a kernel vector of k weights with k rows in the feature vector (representing a k-tokens in the input sentence), producing a new sequence of features. The convolution filter is repeatedly applied to overlapping sections of the matrix, 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. Applying *maximum pooling over time* produces a single output value following each convolution, such that the feature corresponding to a particular vector is the maximum activation in the feature map. Producing a fixed-size output matrix means that variable size sentences and variable size filters can be used to obtaining features with the same output dimensions. This is especially useful for classification tasks, where the desired output is a fixed-sized vector indicating which class the input vector belongs to.

RESULTS

A Evaluation Method

This section discusses our approach to evaluating the successes and failures of the trained models. 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, we 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 [6], therefore we 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.

B News Headlines for Sarcasm Detection

Figure -: F_1 scores of Machine Learning classifiers on the *News Headlines for Sarcasm Detection* dataset

	Vector			
Model Architechture	**BOW**	**TF-IDF**	**GloVe**	**ElMo**
Support Vector Machine	0.714	0.718	0.732	0.844
Logistic Regression	0.721	0.712	0.732	**0.846**
Random Forest Classifier	0.530	0.643	0.754	0.797
Gaussian Naive Bayes	0.662	0.649	0.703	0.640
K-Means	0.400	0.433	0.501	0.471

C Sarcasm Amazon Review Corpus

Figure -: F_1 scores of Machine Learning classifiers on the *Sarcasm Amazon Review Corpus*

	Vector			
Model Architechture	**BOW**	**TF-IDF**	**GloVe**	**ElMo**
Support Vector Machine	0.736	0.775	0.700	0.800
Logistic Regression	0.745	0.748	0.713	**0.813**
Random Forest Classifier	0.525	0.487	0.669	0.727
Gaussian Naive Bayes	0.625	0.621	0.416	0.666
K-Means	0.474	0.334	0.353	0.316

D Experimentation with sentiment features

In this experiment, we strive to answer the research question: *Can we improve the solution by incorporating sentiment features?* We use a state-of-the-art model, called the SentimentAnnotator [21], by Stanford NLP in order to extract features from each of our datasets. Given a snippet of text, the sentiment annotator assigns a discrete label between 0 and 4, where 0 is very negative and 4 is very positive. We contruct 1×6 dimensional vectors for each instance in the dataset, where values 0 - 4 are the proportion of tokens in the snippet that fall into each class, and value 5 is the overall sentiment label assigned to the snippet. For example, the vector [0.1, 0.2, 0.2, 0.1, 0.1, 2] represents a snippet whereby 10% of tokens are very negative, 20% are negative, 20% are neutral, 10% are positive, 10% are very positive and the overall sentiment label is 2 (neutral).

As our features are small, we train a logistic regressor for this task. We can use logistic regression to assess which features are useful for classifying samples. We then use 5-fold cross-validation to compute the F_1 score of the trained models.

Figure -: F_1 score of logistic regression models trained on each dataset

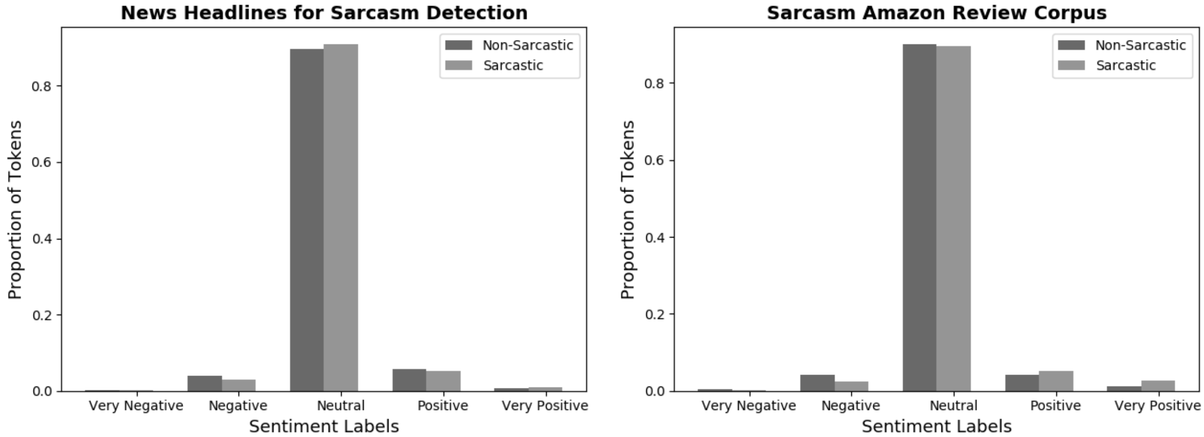
Dataset	**F_1 Score**
News Headlines for Sarcasm Detection	0.515
Sarcasm Amazon Review Corpus	0.691

Our results show that sentiment features improved the model performance when extracted from the Sarcasm Amazon Review Corpus, however when extracted from the News Headlines for Sarcasm Detection, they were ineffective. To further examine this relationship between sarcasm and sentiment, we compute the mean proportion of tokens in each sentiment class for sarcastic and non-sarcastic instances. A visualisation of these results is given in the figure below.

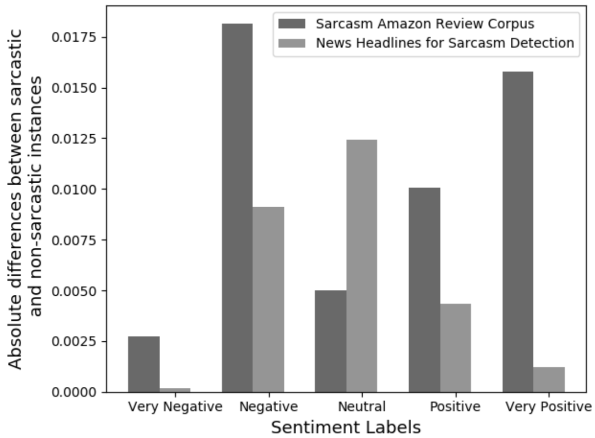
In the figure depicting the results for the Sarcasm Amazon Review Corpus, within most sentiment classes there is a greater difference in the proportion of tokens for non-sarcastic and sarcastic instances. This indicates that sarcastic and non-sarcastic instances are easier to distinguish, given that their sentiment compositions are more distinct. The News Headlines for Sarcasm Detection figure displays the opposite. 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.

Figure -: Absolute differences between the proportion of tokens

Figure -: Mean proportion of tokens in each sentiment class



in each sentiment class between sarcastic and non-sarcastic instances



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

B Solution Limitations

C Lessons learnt

D Project organisation and approach

CONCLUSIONS

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*

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

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] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.