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 powerful linguistic anomaly that when present in text, can alter its meaning entirely. 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, the ultimate aim of this project was to produce a tool that can be used to detect sarcasm with a *high degree* of accuracy.

Method – In my endeavour to realise this aim, I used state-of-the-art contextualised word embeddings and deep learning classification models.

Results – Through extensive experimentation, I found that

Conclusions – Following this experimentation, I conclude that

This section should not be longer than half of a page, and having no more than one or two sentences under each heading is advised. Do not cite references in the abstract.

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

INTRODUCTION

Sarcasm is defined as the use of remarks that clearly mean the opposite of what they say, made in order to hurt someone's feelings or to criticize something in a humorous way [16]. Sarcasm poses a significant challenge within the field of natural language processing, specifically within sentiment analysis, as it transforms the sentiment polarity of a positive or negative utterance into its opposite. 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 recognize sarcastic intent; due in part to the lack of an all-encompassing, universal definition of sarcasm. In a 2011 study, González-Ibánez et al. [6] 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. If we could replicate this performance, or even perhaps improve upon it, we could 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 –

Which linguistic cues indicate sarcastic intent in written text? How can a model be used to detect the words that correlate more to sarcastic labels? Do deep learning techniques perform better than machine learning approaches for sarcasm detection? Can the solution be used to improve the sentiment analysis task? Does the proposed solution perform well on other datasets?

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

The **minimum** objectives of this project were to evaluate, compare and clean high-quality datasets, as well as to experiment with and evaluate machine learning architectures on these datasets. I achieved this by ...

The **intermediate** objectives of this project were to experiment with and evaluate deep learning models on the chosen datasets, and determine which is the best performing solution. Additionally, I also evaluated the model against unseen data. I achieved this by...

The **advanced** objectives of this project were 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. I achieved this by ...

RELATED WORK

In this section, we will 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 as they may have potential to perform well in this specific domain. 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 [21]. Hence, 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.

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. [15]
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. [16]

A Traditional Classifiers

A number of simple linguistic approaches have been used in previous research into sarcasm detection. One such class of naive approaches is *rule-based*, where text is classified based on a set of linguistic rules. Maynard and Greenwood (2014) [10] proposed that the sentiment contained in hashtags can indicate the prescence of sarcasm in tweets, such as #notreally in the example "I love doing the washing-up #notreally". They used a rule-based approach to determine that if the sentiment of a tweet contradicts that of the hashtag, then this tweet is an example of sarcasm. In Riloff et al. (2013) [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 sentence. 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

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

Tsur et al. (2010) [21] 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 utilizes 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 N.B. neighbours are vectors that share at least one pattern feature. SASI was evaluated against a gold standard dataset of unseen sentences - achieving a precision of 77% and recall of 83.1%, resulting in an F_1 score of 79.9% on newly discovered sentences.

Naive Bayes classifiers are supervised algorithms that use bayes theorem to make predictions. Reyes et al. (2013) [18] used a naive bayes and decision trees algorithm in order to detect irony which is closely related to sarcasm. They experimented with balanced and unbalanced distributions (25% ironic tweets and 75% other), achieving an F-score of 0.72 on the balanced distribution, dropping to 0.53 for the inbalanced distribution. In a similar vein, Barbieri et al. (2014) [1] used random forest and decision tree classifiers, also for the detection of irony. They used six types of features in order to represent tweets, and recorded results over three categories of training data - education, humor and politics.

Support Vector Machines (SVM), can be effective when smaller amounts of training data are available, however they require more computational resources than naive bayes. SVMs use kernel functions to draw hyperplanes dividing a space into subspaces. González-Ibánez et al(̇2011) [6] used two classifiers, support vector machine with sequential minimal optimization, as well as logistic regression. Their best result was an accuracy of 0.65, achieved with the combination of support vector machine with sequential miniminal optimization and unigrams. *Ptáček et al. (2014)* [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 [14,24]. Two common DNN architectures are Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). They typically require a lot more training data than machine learning algorithms, but have been shown to produce state-of-the-art results in several domains.

https://arxiv.org/pdf/1703.03091.pdf

This has already been used Write about RNN uses

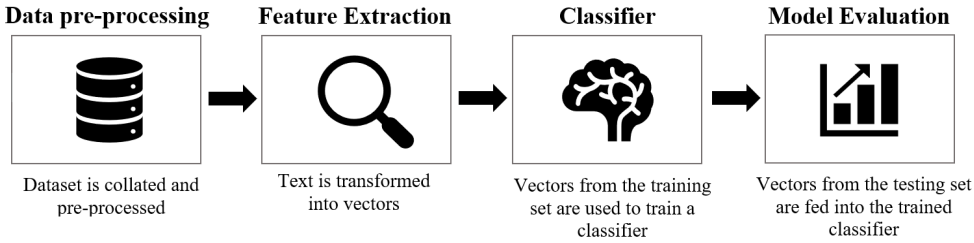
One particularly interesting technique is the *Hierarchical Attention Network (HAN)* defined in Yang et al. (2016) [22] - an attention-based LSTM. The intuition is that certain words and sentences in a document contribute more to its overall meaning, and this is highly context dependent. They included 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)* [5] 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)* [23] 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)* [24] 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)* [14] describes a convolutional neural network (CNN) for sarcasm detection.

SOLUTION

This project addresses a classification problem. In the simplest terms, this project aims to identify, given a snippet of text: is it sarcastic? or is it 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 out

Figure -: Overview of the model pipeline



A Implementation Tools

This project was majoritively written in python, as it has an abundance of open-source libraries well-suited to machine learning and deep learning tasks. We used **pandas** for manipulating the datasets, as it provides functionality for applying functions globally across the dataset. This simplifies the process of data pre-processing. We also incorporated **SpaCy** when cleaning and tokenizing the data. In terms of classification, I did most of my machine learning implementation using **scikit-learn**, and used **keras** to construct deep learning models. The model interface was written using web-application development languages including HTML, CSS and JavaScript.

B Datasets

I used publically available datasets in order to produce results. I avoided using datasets of tweet ids, whereby the original tweet must be scraped using the Twitter API. This is because some of the original tweets are non longer available on twitter, therefore this would not make for a fair comparison between my results and the results of the source study. I experimented with data from multiple domains and online platforms, including Twitter, Amazon, Reddit and even news headlines. This data was all user generated, although data from news headlines is more likely to be well-structured than data from more informal platforms.

In News Headlines, I expect that the sarcasm may be more contextual and that the data will be free from spelling and grammar errors.

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

Dataset Title	Number of data points	Avg. #tokens	% in GloVe dictionary
News Headlines Dataset For Sarcasm Detection	28619 +ve: 47.6% -ve: 52.4%	11.2 tokens	97.8%
SARC	1010826 +ve: 50% -ve: 50%	10.46 words	
Sarcasm Amazon Review Corpus	1254 +ve: 34.9% -ve: 65.1%	276.4 tokens	98.2%

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 capitalization, 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. I decided to 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)* [9] showed that data contained in hashtags can be used to indicate sarcasm, such as #not. Transforming text into lowercase is a common pre-processing strategy, useful for reducing sparsity caused by variations in letter capatiliztion e.g. 'Cat', 'cAt', and 'CAt' are all mapped to the same word - 'cat'. Similarly, removing punctuation and extending contractions (e.g. don't $\rightarrow$ do not) is commonplace. However, punctuation and capitalization can be used for emphasis, therefore removing these attributes may make the presence of sarcasm less obvious.

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 Vectorization of textual data

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

D.0.1 Traditional Approaches – Perhaps the simplest vectorization approach is *Bag of Words (BOW)*, which encodes text as a single vector containing a count of the number of occurrences of each word in the snippet. *Term Frequency-Inverse document frequency (TF-IDF)* [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. Predictably, vectors produced in this way do not preserve the context within which words are found, complicating the task of discerning their meaning. Hence, these approaches are unlikely to be suitable in the application domain of detecting sarcasm - a highly contextual phenomenon. The *Bag of N-grams* approach is a generalisation of Bag of Words, whereby instead of counting the number of occurrences of each unigram, we count the number of occurrences of each N-gram. This allows us to capture a small window of context around each word, however this results in a much larger and sparser feature set.

D.0.2 Word Embedding – First introduced in Mikolov et al (2013a) [11], *Word2Vec* describes a group of related models that can be used to produce high-quality vector representations of words - two such models are *Continuous Bag-of-Words* and *Continuous Skip-Gram*. It consists of a shallow (two-layer) neural network that takes a large corpus and produces a high-dimensional vector space. Unlike previous techniques, words that share a similar context tend to have collinear vectors, that is to say they are clustered together in the feature space. Consequently, Word2Vec is able to preserve the semantic relationships between words, even constructing analogies by composing vectors e.g. king - man + woman $\approx$ queen. Likewise, it captures syntactic regularities such as the singular to plural relationship e.g. cars - car $\approx$ apples - apple. In Word2Vec, intrinsic statistical properties of the corpus, which were key to earlier techniques, are neglected, therefore global patterns may be overlooked. To mitigate this, the *GloVe* [12] approach generates word embeddings by constructing an explicit word co-occurrence matrix for the entire corpus, such that the dot product of two word embeddings is equal to log of the number of times these words co-occur (within a defined window).

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

D.0.3 Contextualized Word Embedding – In pursuit of a more robust approach, we look to deep neural language models. Peters et al (2018) [13] introduced the *ELMo* model, and showed that the addition of ELMo to existing models can markedly improve the state-of-the-art in a number of NLP tasks. ELMo utilizes a bi-directional LSTM *(long short-term memory)* model, concatenating the left-to-right and right-to-left LSTM in order to produce deep contextualised word representations. Like fastText, they are character based, therefore robust representations can be constructed for out-of-vocabulary tokens. However, rather than 'looking-up' pre-computed embeddings, ELMo generates them dynamically. Hence, it can disambiguate the sense of a word given the context in which it was found.
Devlin et al. (2018) [4] introduced the *BERT* framework, a multi-layer bidirectional transformer model consisting of two main steps - *pre-training* and *fine-tuning*. During pre-training, unlabeled data is used to train the model on different tasks, providing some initial parameters that are tweaked in the fine-tuning stage (a procedure known as *transfer learning*). BERT uses a *masked language model* which "masks" 15% of the input tokens with the aim of predicting them based on their context. This allows BERT to leverage both left and right contexts simultaneously, unlike ELMO which uses shallow concatenation of separately trained left-to-right and right-to-left language models.

I experimented with feature extraction techniques from each category - for word embeddings, I used GloVe, and for contextualised word embeddings I experimented with ElMo.

E Classification

Observing that people are inclined to be more sarcastic towards specific subjects such as weather or work. Feed features to trainer such as support vector machine. Represent input words numerically. One-hot encoding has two main disadvantages, firstly, each vector is the size of the vocabulary, with a single 1 in the column marking the word. This creates sparse inefficient representations.
There exists an extensive body of literature covering machine learning and deep learning approaches to natural language processing problems, however in the application domain of sarcasm detection, they mostly display low accuracy.
I experimented with a number of combinations of feature extraction techniques and classification models. A summary is given below:

E.1 Machine Learning Models

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

E.1.1 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.

E.1.2 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.

E.1.3 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 Naive Bayes - including Multinomial, Bernoulli and Gaussian.

E.1.4 Logistic Regression – A statistical model commonly used for binary classifcation. It 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.

E.1.5 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

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 [8] instead which are more effective at preserving long term dependencies.

E.2.2 Convolutional Neural Networks – Convolutional neural networks output fixed sized vectors, hence they are often applied to classification tasks. Convolutions are very fast.

Convolutional Neural Networks have sought success in a number of tasks

They consist of neurons with weights and biases CNNs consist of a few layers of convolutions with non-linear activation functions - they have fewer layers than typical neural networks. Each layer applies dfferent filters and combines their result.

Pooling layers are used, typically after convolutions. They subsample their input. Usually, performing pooling consists of applying a max operation to the result. Pooling provides a fixed size output matrix, typically required for classification. Allows variable size sentences and variable size filters, obtaining same output dimensions.

Each row of a matrix corresponds to one token Filters slide over full rows of the matrix of the input matrix., therefore the width of filters is typically the widt

RESULTS

A Evaluation Method

This section will discuss 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, I will instead form a conclusion based on the F_1 score i.e. the harmonic mean of precision and recall, where scores range from 0 (worst) to 1 (best). This metric has faced some criticism for giving equal weight to both precision and recall [7], therefore I will consider both measures separately, as well as in combination.

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

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

Figure -: Results of Machine Learning Classifiers on the Sarcasm Amazon Review Corpus

Model architecture	F_1 **Score**
GloVe + Support Vector Machine	0.739
GloVe + Logistic Regression	0.730
GloVe + Random Forest Classifier	0.644
GloVe + Gaussian Naïve Bayes	0.415

B Sentiment Analysis Experimentation

In this experiment, we strive to answer the research question: *Can the solution be used to improve the sentiment analysis task?* We hypothesise that incorporating a sarcasm label as an additional feature for the sentiment analysis task will improve the performance of a sentiment analyser. I used a pre-built state-of-the-art model, and a gold-standard benchmark dataset against which to test my hypothesis.

EVALUATION

In the following section, we will 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

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

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

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

[6] 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.

[7] 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.

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

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] Cambridge University Press. Irony: meaning in the cambridge english dictionary. `https://dictionary.cambridge.org/dictionary/english/irony`, 2020. Last accessed: 11 Jan 2020.

[16] Cambridge University Press. Sarcasm: meaning in the cambridge english dictionary. `https://dictionary.cambridge.org/dictionary/english/sarcasm`, 2020. Last accessed: 11 Jan 2020.

[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] 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.

[22] 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.

[23] 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.

[24] 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.