Sarcasm Detection: How Sub-LIME!

Group ID : 32
Group members:
Rahul Trada, student number: 1018650
Michail Koupparis, student number: 14062973
Marilena Lemonari, student number: 19045031
Myrto Papakonstantinou, student number: 110025679
March 2020

Abstract

Sarcasm detection is a narrow research field in Natural Language Processing (NLP), and can be considered a specific case of sentiment analysis, where the task is to detect if a given sentence is sarcastic or not. The challenge is that, unlike the general case of sentiment analysis where words imply distinct sentiment (e.g. 'love' is positive, 'hate' is negative), sarcasm is not so clearly defined, and even humans have difficulty detecting it sometimes. We investigate the sarcasm detection performance of four model architectures, namely the vanilla Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU) and Convolutional Neural Network (CNN) on a news headlines dataset. We further employ the Local Interpretable Model-Agnostic Explanations (LIME) tool on sarcastic headlines to explore potential differences in model explanations. Results show that the two best performing models on the task are the LSTM and CNN and that all models tend to provide similar explanations for sarcasm classification.

1 Introduction

Sarcasm or verbal irony is a form of speech people use to be critical or funny. By its nature, sarcasm is inherently ambiguous and even humans might have difficulty grasping it at times (Sivaprakasam and Jayaprakash, 2017). Sarcasm has even been shown to vary between cultures (Joshi et al., 2016) and relies heavily on cues such as facial expressions, intonation, context and sometimes a familiarity with the person expressing it. Many of these components are absent in text, making sarcasm detection a challenging task. Furthermore, automatic sarcasm detection is important in opinion mining, sentiment analysis and advertising (Liu et al., 2014). There has been a lot of progress and demand for machine learning models in these fields. This motivates us to explore the best models for this task and understand their behaviour.

Our present aim is to compare the performance of deep neural networks, specifically RNNs, LSTMs, GRUs and CNNs, on the task of sarcasm detection. We further intend to analyse the behaviour of the models in order to explain their relative performance using the black-box local explanation method, LIME (Ribeiro et al., 2016). We would also like to draw parallels between the features the models consider important, and those that humans use in detecting sarcasm. We hypothesise that CNNs will outperform other architectures for this task due to them being well suited at extracting position-invariant features. On the other hand, RNNs are better at modelling long-range dependencies in text due to their sequential structure. However, given our dataset consists of news headlines, which tend to be short, we do not expect RNNs to have much of an advantage.

The outcome of our work is that the LSTM and CNN architectures have the best performance but any difference between the two is not statistically significant. Furthermore, the models tend to provide similar explanations for sarcasm classification. However, these explanations vary in how they align with humans' interpretation – that is, sometimes they agree with human intuition and other times not.

2 Related Work

Earlier work in automatic sarcasm detection relied on manual feature engineering because of the use of discrete classifiers such as Support Vector Machines (Baktha and Tripathy, 2017; Burfoot and Baldwin, 2009). However, the ability of neural networks to automatically extract important and subtle features has reduced the need for heavy feature engineering and has made them the gold standard for most NLP tasks. Therefore, in this study we only considered neural network models.

Existing literature has proven the superiority of recurrent neural networks (vanilla RNN, LSTM, GRU) over discrete models in both sentiment analysis and specifically sarcasm detection (Baktha and Tripathy, 2017; Porwal et al., 2018; Zhang et al., 2016). Zhang et al. (2016) found that the use of contextual features (historical author tweets) in addition to local features was effective in boosting the performance of their bi-directional gated recurrent neural network for sarcasm detection. We were only able to use local features in our study due to the nature of our dataset (Section 4.1) as we could not connect the author with the headlines.

Furthermore, CNNs are a class of deep learning models that have found great success in computer vision tasks, primarily because they are well suited at detecting position-invariant spatial hierarchies of features in the input (Yamashita et al., 2018). Moreover, work on CNNs has demonstrated that they can achieve state-of-the-art results on multiple NLP benchmarks, as well as the effectiveness of using pre-trained word embedding vectors on various classification tasks (Kim, 2014), such as sentiment analysis (dos Santos and Gatti, 2014). Poria et al. (2016) achieved cutting-edge results on sarcasm detection on a dataset of tweets, using a novel CNN architecture that combines multiple networks individually trained to detect sentiment, emotion and personality features. We did not incorporate such features due to the lack of contextual information in our dataset. However, we employed the Google News Word2Vec embeddings for our tasks in this paper (Section 4.5).

Due to the difference in the structures of RNNs and CNNs, we speculated that CNNs are better at tasks where sequence ordering is not so important, such as sentiment analysis, and RNNs in cases where long-term dependencies in sequences are crucial, such as language modeling. However, the literature does not support such a clear distinction between the two. A comparative study on the performance of CNNs and RNNs on a variety of NLP tasks found that the two architectures have similar performance in text classification tasks and that hyperparameter tuning may play a greater role in improving performance (Yin et al., 2017).

To the best of our knowledge, there has not been a direct comparison of CNN and RNN based models for the specific task of sarcasm detection, which is what we investigate here.

3 Methodology

For this classification task, we trained four models, three RNN variants (vanilla RNN (Elman, 1990), LSTM (Hochreiter and Schmidhuber, 1997), GRU (Chung et al., 2014)), and a CNN (LeCun et al., 1998). The RNN variants share several properties as they all have a single-cell architecture, process inputs sequentially and operate in a loop as the output is fed back into the cell until the last time-step. However, differences between these variants exist in the inner mechanics of the cell as described in the following sections (Sections 3.1, 3.2, 3.3). Further, we describe the CNN architecture in Section 3.4. To interpret our models we used the LIME tool (Ribeiro et al., 2016), which we elaborate on in Section 3.5.

3.1 RNN

The inputs to the vanilla RNN cell are the previous state, s_{t-1} and the incoming input of the sequence, x_t. At $t = 1$, the initial state, s_0, is typically initialised to zero. These inputs are concatenated, multiplied by a randomly initialised weight matrix and added to a bias vector. The $tanh$ activation function is applied to this result as it regulates the output of the neural network ensuring the output values are between -1 and 1. This operation results in the output, h_t, and next state, s_t. The next state s_t gets fed into the cell again merging with the input at the next time-step to perform the same operation again. Depending on the task, the relevant output/s are used at the appropriate time-step. In our case, the task is a many-to-one model where, given an input sentence (news headline), we use the last hidden state – a dense vector representation of the entire sequence – to pass into a fully connected layer for sarcasm classification.

With longer sequences, the latter RNN states start to lose the ability to retain important information present in earlier time-steps. RNNs also suffer from the vanishing and exploding gradient problems. These limitations gave rise to the more advanced LSTM and GRU architectures.

3.2 LSTM

The LSTM is an advanced version of the vanilla RNN which solves the short-term memory problem using the concept of gates. For LSTMs, the inputs of each cell are the previous cell state c_{t-1}, the previous hidden state h_{t-1} and the current state x_t. The purpose of cell states is to transport relevant information through the cells. This is done with the help of three gates namely, the forget, input and output gate. Firstly, the previous hidden state h_{t-1} and the current state x_t are concatenated and passed into the forget gate (output is f_t). In the forget gate, a sigmoid function is applied to the concatenated vector. Sigmoid functions have range $[0, 1]$ helping to keep relevant information (value closer to 1) and get rid of irrelevant information (value closer to 0). For the input gate, the concatenated vector of the previous hidden state and current state is passed through a sigmoid function (output is i_t) and also a $tanh$ function and the two outcomes are multiplied together and then added to the product $h_{t-1} * f_t$. This gives the next cell state c_t. Finally, for the output gate, the concatenated vector is again passed through a sigmoid function (output o_t) and after the new cell state is passed through a $tanh$ function, the two outcomes are multiplied together giving the new hidden state h_t which will be transferred to the next cell (next time-step).

3.3 GRU

Similarly to the LSTMs, GRUs use gates to solve the short-term memory problem. The GRUs have two gates, namely the reset and update gate. The former decides which information to forget and the latter which information to add or throw away. For the reset gate, the concatenated vector of the previous hidden state h_{t-1} and current state x_t are passed through a sigmoid function (operation output r_t). For the update gate, the concatenated vector is again passed through a sigmoid function (operation output z_t). The previous hidden state and the output of the reset gate are multiplied together ($h_{t-1} * r_t$), concatenated with the current state x_t and passed through a $tanh$ function (to give s_t). Then the output of the update gate is multiplied by z_t and added to the product $h_{t-1} * (1 - z_t)$. The output of this operation is the next hidden state $h_t = s_t * z_t + h_{t-1} * (1 - z_t)$.

3.4 CNN

CNNs utilise convolving filters to detect local features at various positions of the input. Different filters will detect different features, and this process can be repeated several times, resulting in a hierarchy of feature extraction, from low-level features in the lower layers, to higher-level features in the higher layers.

Suppose we have an input sequence with *n* tokens. The input will then have size nxd. This can be thought of as the result of a concatenation of n 1xd vectors, where *d* is the embedding dimension. Let $x_{i:j}$ refer to the concatenation of tokens $x_i, ..., x_j$ in the input. The convolution operation works as follows: We have a filter *w* (of size hxd, where *h* is the size of the token window $x_i, ..., x_j$). The convolution operation is $c_i = f(w \cdot x_{i:j} + b)$, resulting in an output number c_i. Here *f* is some nonlinear activation function, such as ReLU, and *b* is a bias term. This convolution operation is performed on every possible window of size h (h-grams) over the input, resulting in a vector of outputs $[c_1, ..., c_{n-h+1}]$. Note we will be using filters of different sizes, to extract features from n-gram representations for different *n* values, resulting in vector outputs of varying sizes. These vectors can be thought of as different characteristics of the input. We then perform a max pooling operation over these vectors, to extract the most important feature, resulting in a single output scalar per filter. These scalars are then concatenated to form a vector of size (*number of filters per kernel size* x *number of different kernel sizes*) that will serve as input into a multi-layer perceptron for classification (Kim, 2014).

3.5 Analysis-LIME

LIME is a tool used to investigate the behaviour of machine learning models and understand possible drawbacks (Mary, 2019). It aims to make black-box models, such as neural networks, more transparent and interpretable by providing an insight into local explanations for specific instances. In our case, it can help identify how each feature-word presented in a sentence influences the model to classify the whole sentence as sarcastic or not. We wanted to examine the behaviour of the models and determine if the model predictions are justified. We also wanted to explore the similarity of the models' explanations and if they resemble human intuition.

LIME can provide qualitative representations as the ones shown in Section 5.2. Qualitative representations can help researchers understand models better and find potential pitfalls rather than just focusing on the evaluation results (Ribeiro et al., 2016).

LIME treats the model as a black-box (e.g has

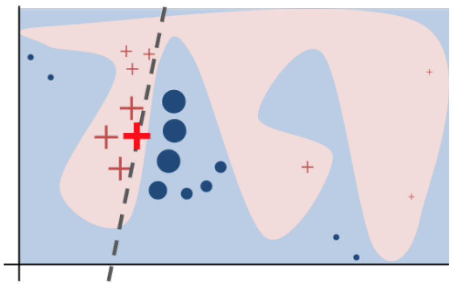


Figure 1: LIME's mechanism described by Ribeiro et al. (2016):
" The black-box model's complex decision function f (unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bold red cross is the instance being explained. LIME samples instances, gets predictions using f, and weighs them by the proximity to the instance being explained (represented here by size). The dashed line is the learned explanation that is locally (but not globally) faithful. "

no access to its parameters) and feeds through it different similar permutations of the input and observes how the prediction changes as shown in Figure 1.

4 Experiments

4.1 Data

We used the News Headlines dataset (Misra and Arora, 2019) from Kaggle[1] for our sarcasm detection experiments. Sarcastic news headlines were collected from *TheOnion*, a website that puts a satirical spin on current events. Non-sarcastic news headlines were collected from *HuffPost*. This dataset has several advantages over the typical social media (mainly *Twitter*) datasets used for sarcasm detection. For instance, these headlines are free from spelling mistakes and slang, making the use of pre-trained word embeddings more feasible. Moreover, the headlines are self-contained, reducing the noise which is frequent in tweets that are part of a thread.

4.2 Prepossessing

We used the Keras tokenizer[2] on the training corpus, and set a limit on the size of the vocabulary (maximum features = 50,000). We capped the vocabulary in this way to reduce the number of learnable parameters and as a result the chance of overfitting. Moreover, the variable-length sequences were padded according to the longest sequence in the training set in order to batch sequences together and thus efficiently perform the necessary matrix operations. We carried out a preprocessing experiment with bigrams in addition to the standard unigrams but did not observe an improvement in performance for the RNN variants. This could have been due to the increase in learnable parameters, making the model unnecessarily complex and more prone to overfitting. This result, coupled with the use of pre-trained word embeddings, motivated our decision to adhere to unigrams for the RNN experiments.

4.3 Training

We trained the models using Supervised Learning and minimising the Binary Cross Entropy loss (Equation 1). Additionally, we used the Adam optimiser which is an adaptive learning rate optimisation algorithm making use of momentum and weight decay (Kingma and Ba, 2014).

$$L = -\sum_i (y_i log(p_i) + (1 - y_i) log(1 - p_i)) \quad (1)$$

4.4 Evaluation

The News Headlines dataset is composed of 28,619 headlines. We split the data randomly, keeping the proportion of sarcastic and not sarcastic data consistent in each set. The dataset breakdown was as follows: Train-19,460, Validation-4,866, Test-4,293. The validation set was used throughout training to tune hyperparameters and prevent overfitting via early stopping. We employed early stopping at 15 epochs, i.e. we stopped training if the validation loss had not decreased in the last 15 epochs.

4.5 Hyperparameters

The hyperparameter tuning was a combination of grid and manual search. The hyperparameters tuned were learning rate, batch size, dropout rate and classifier architecture (hidden layer number and size). Some model-specific hyperparameters were bidirectionality and number of RNN layers for RNN variants and filter sizes for CNNs. For all RNN variants, it was interesting to note that the parameters that yielded the lowest validation loss were the same. The best parameters for all models are shown in Table 1.

[1] Sarcastic News Headlines Dataset
[2] Keras Tokenizer

Hyperparameter	**Values**			
	RNN	**LSTM**	**GRU**	**CNN**
No. Hidden Layers	1	1	1	3
Hidden Layers Units	[128]	[128]	[128]	[64,32,16]
RNN Layers	1	1	1	-
Bidirectional	False	False	False	-
Filter Sizes	-	-	-	[1,2,3,4,5,10,20]
Learning Rate	0.001	0.001	0.001	0.001
Batch size	1024	1024	1024	1024
Dropout	0.5	0.5	0.5	0.5

Table 1: Best Hyperparameters

Another hyperparameter was the use of pre-trained word embeddings. We used Google's pre-trained Word2Vec embeddings (Mikolov et al., 2013), 300-dimensional dense vector representations of words trained on about a billion words from the Google News dataset. Consistent with the literature, we found that initialising the embedding layer with these pre-trained word embeddings boosted performance for all models. We further concluded that updating these embeddings during training to fine-tune them for our sarcasm detection task was beneficial to performance. As is a common practice, the embedding lookup table was downsampled to include words from only the training corpus/vocabulary. Furthermore, we experimented with initialising any unknown words randomly from a normal distribution or with the 'UNK' vector. Initialising with the 'UNK' vector yielded better results for all models.

4.6 Quantitative LIME Analysis

In addition to the qualitative LIME analysis, presented in Section 5.2, we also performed further quantitative analysis on LIME results to evaluate how similarly our models behave. To accomplish this, we used cosine similarity which determines whether two vectors are pointing in roughly the same direction. We deemed this an appropriate similarity measure due to its use in various text tasks such as semantic similarity in word embeddings (Li and Han, 2013). Moreover, we wanted to further examine how this differs when we only consider sarcastic instances, non-sarcastic instances and when both are included. We also explored how often different models have different explanations for their predictions. In order to get explanations we used LIME (Section 3.5) for 500 randomly chosen examples from the test set, each time based on label and then filtered based on the prediction of the models. We carried this out for each pair of models and used a threshold of 0.85. This means that for any value above the threshold, we considered the pair of models to have the "same" explanation and thus "same" behaviour. Also, we recorded the average cosine similarity to better understand how close the behaviour of each pair is. We present our results in Section 5.3.

4.7 Stop Words

In an attempt to assess the significance of stop words for sarcasm detection, we removed them from the data and repeated the experiment. This change was made before the training/ test split and before tokenisation and padding. Next, the four models were trained with this different preprocessing step using the best hyperparameters found from the initial round of experiments.

4.8 Different Data Domain

As part of our experiments, we evaluated our trained models against a social media domain for the same task to test how well our trained models could generalise. This dataset from Kaggle was based on Reddit comments (SAR, 2017). Given the magnitude of the dataset (1.3 million comments), a very downsampled proportion (0.5%) was used in this experiment.

5 Results and Discussion

5.1 Model Performance

Model	Loss	Accuracy	F1-Score
RNN	0.3576	84.93%	0.8434
LSTM	0.3136	**86.89%**	**0.8608**
GRU	0.3351	85.91%	0.8486
CNN	**0.3036**	86.63%	0.8543

Table 2: Model Test Set Metrics

Table 2 summarises the performance of our models with the best parameters on the test data. The best performing models across all metrics were the LSTM and the CNN. We gave particular importance to the loss criterion which does not take into account just the correctness of a prediction, but its confidence (probability) too. Moreover, Figure 2 gives us an insight into the individual model predictions. We observed that, with the exception of the vanilla RNN model, all other models tended to give more false positives rather than false negatives.

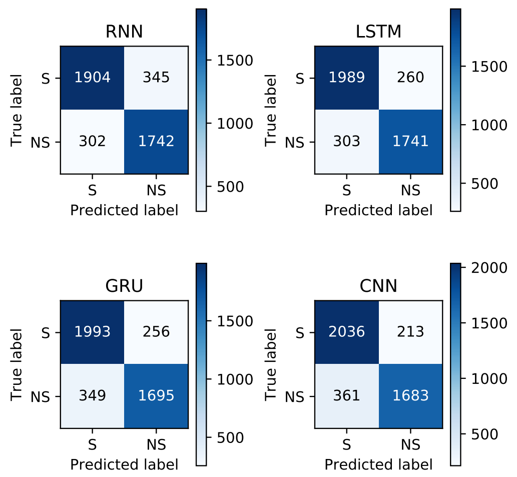


Figure 2: Test Set Confusion Matrices

5.1.1 Statistical significance

Given the difference in performance between the LSTM and CNN was quite small as shown in Table 2, we decided to investigate if the observed difference was statistically significant. As we are dealing with a binary classification problem, the appropriate statistical test to compare two models is the McNemar's test (McNemar, 1947), which uses a contingency table of the accuracy of the two models' predictions.

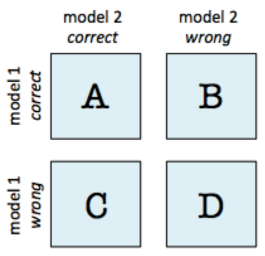


Figure 3: Contingency table

Given a contingency table as in Figure 3, the McNemar (chi-squared) test statistic is computed as follows:

$$\chi^2 = \frac{(|B-C|-1)^2}{B+C} \qquad (2)$$

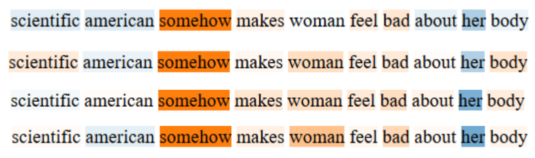
This test compares the frequency of predictions that the LSTM got correct while the CNN got wrong, and vice versa, to obtain a chi-squared statistic with 1 degree of freedom. For our models, the contingency table has the values shown in Table 3.

	CNN correct	CNN incorrect
LSTM correct	3510	180
LSTM incorrect	209	394

Table 3: LSTM/CNN Contingency Table

The null hypothesis is that none of the models performs better than the other, with the alternative hypothesis being that the performances of the two models are not equal. Using a relevant statistical package (mlxtend[3]) in Python, we obtain a chi-squared statistic of 2.015, with a corresponding p-value of 0.15, which, when using a significance level α of 0.05 or less, indicates that there is insufficient evidence to reject the null hypothesis. Therefore, we conclude that none of the models performs better than the other.

5.2 LIME Qualitative Analysis Results


scientific american somehow makes woman feel bad about her body

scientific american somehow makes woman feel bad about her body

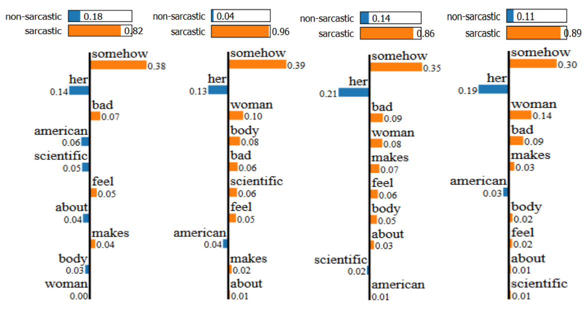
scientific american somehow makes woman feel bad about her body

scientific american somehow makes woman feel bad about her body

Figure 4: LIME Example 1

We selected several instances from the test set and evaluated them under the LIME microscope to get an insight into local explanations for predictions and identify potential differences between our models. From Example 1 (Figure 4 and Figure 5), we can see that all models correctly predicted the sentence as sarcastic and identified the word "somehow" as the strongest feature indicating sarcasm. Despite there being no differences in explanations between models, it was interesting to note that this common explanation is in line with what

[3]Mlxtend Package



(a) RNN (b) LSTM (c) GRU (d) CNN

Figure 5: LIME Example 1

we would expect humans to select as the main sarcastic component of the sentence. In Example 2 (Figure 6 and Figure 7), all models once again correctly predicted the sentence as sarcastic for similar reasons. However, humans would agree that the characteristic that makes this sentence sarcastic is “83rd”. This was not picked up by any model, and was conversely assigned as a feature for the non-sarcastic class, indicating the limitations of these models. That being said, it may be unrealistic to expect a model to detect these nuances given the size of our training dataset. Overall, none of the individual test instances we explored showed drastic differences in explanations between models. We tested this more rigorously in the next section (Section 5.3).

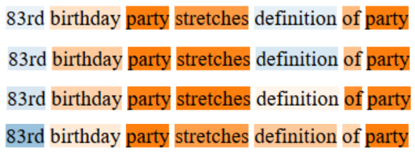

83rd birthday party stretches definition of party
83rd birthday party stretches definition of party
83rd birthday party stretches definition of party
83rd birthday party stretches definition of party

Figure 6: LIME Example 2

5.3 LIME Quantitative Analysis Results

Table 7 summarises the results from the experiment described in Section 4.6. As can be seen from the table the explanations we obtained from LIME for each pair are quite similar. We observed that the order of similarity is RNN-CNN << RNN-GRU <= RNN-LSTM << GRU-CNN << LSTM-CNN << LSTM-GRU. Also, we noticed that when we take into consideration only sarcastic instances for all models there is a drop in similarity while there is an increase when we picked only non-sarcastic. This implies that all model pairs have more similar behaviours when detecting non-sarcastic instances. Moreover, it can be seen that the results are fairly close even if you consider only instances when the two models have the same prediction and when this is correct. This supports our qualitative analysis results in Section 3.5 since the average cosine similarities vary from 0.810 to 0.953 of a maximum of 1.

5.4 Stop Words

LIME analysis revealed that sometimes stop words carry substantial weight, influencing the classification decision (Figure 10 and Figure 11). This contradicts human intuition, motivating the removal of stop words experiment. After removing and re-training, looking back at the example that inspired this experiment, we can see that removing stop words steered the models towards correctly predicting the sentence as sarcastic (Figure 12 and Figure 13).

Model	Loss	Accuracy	F1-Score
RNN	0.4625	78.83%	0.7684
LSTM	0.4109	82.30%	**0.8157**
GRU	0.4185	81.23%	0.7829
CNN	**0.4053**	**82.88%**	0.8124

Table 4: Model Test Set Metrics - No Stop Words

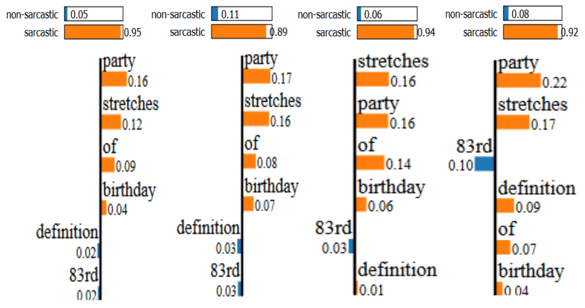
Table 4 shows the results of this experiment on the entire test set. The LSTM and CNN are the two best models, a result consistent with findings obtained in Section 5.1. However, we note that excluding stop words did not improve the models' overall performance. Repeating the hyperparameter tuning process using this preprocessing step, might give better results or it could be the case that stop words are important for grasping the meaning of a sentence and therefore classifying it correctly.

5.5 Different Dataset Domain

Model	Loss	Accuracy	F1-Score
RNN	1.1472	47.56%	0.3494
LSTM	1.3863	49.14%	0.2908
GRU	1.3643	49.22%	0.2860
CNN	1.2264	48.72%	0.2752

Table 5: Model Metrics - Reddit Dataset

As clearly illustrated in Table 5, the experiment conducted using the Reddit dataset yielded less than favourable results with high losses and accuracies lower than chance prediction, 50%. We ex-



(a) RNN (b) LSTM (c) GRU (d) CNN

Figure 7: LIME Example 2

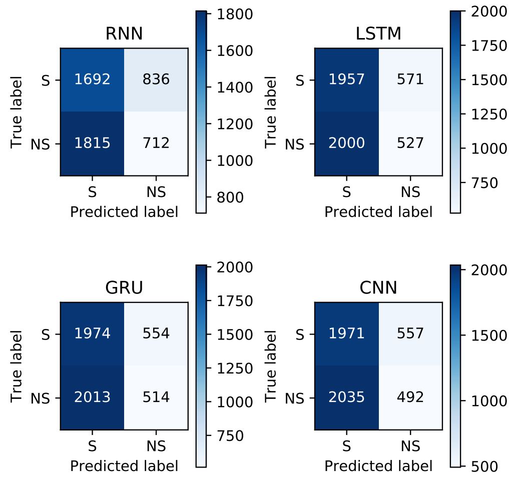


Figure 8: Confusion Matrices - Reddit Dataset

pected a drop in performance due to inherent differences in the News Headlines dataset we trained on and the Reddit comments dataset. News headlines are self-contained, incorporating the sarcasm within the sentence, whereas Reddit comments are witty retorts to an existing "parent" comment. Moreover, the Reddit dataset was more noisy due to informal use of language and higher variability in style and length of responses. However, the discrepancy observed in model performance was too significant and thus could be a result of our limited training dataset or simply a lack of generalisability of our models.

Interestingly, Figure 8 shows that all models overwhelmingly classified Reddit comments as sarcastic. This could be just by chance or it could be that the style of Reddit comments aligned with sarcastic features learned and extracted by our trained models.

6 Conclusion and Future Work

We set out to explore the performance of four model architectures, RNN, LSTM, GRU and CNN on the challenging task of sarcasm detection. We found that the best performing models were the LSTM and CNN. However, we cannot conclude that one model was superior to the other since the differences in the test metrics between the two was not found to be statistically significant. Consequently, our results do not strictly support our initial hypothesis that CNNs would outperform the RNN variants on this task. We placed these models under further scrutiny using the LIME tool. The main outcome of our LIME experiments was that the four models do not significantly differ in the explanations they provide for classification of sarcastic and non- sarcastic headlines.

Some limitations of our work include the relatively small dataset and the lack of contextual features such as the topic of headlines or information about the author such as style of writing or personality traits. Moreover, our models did not include an attention-based component because our primary goal was to compare the models rather than achieve state-of-the-art results on the task.

However, work on attention and the transformer architecture (Vaswani et al., 2017) have taken the NLP community by storm, setting new benchmarks in performance across a wide variety of tasks. More specifically, Potamias et al. (2019) have used transformer-based models on irony and sarcasm detection with promising results. The main advantages of transformer models, which completely rely on attention mechanisms, over traditional sequential RNNs is their ability to parallelize and thus increase memory and computational efficiency (Vaswani et al., 2017). For classification, the transformer architecture is presented in Figure 9. We wanted to explore this ourselves and tried a pre-trained BERT model (Devlin et al., 2018) which yielded favourable results for the News Headlines dataset but subpar results for the Reddit dataset (Table 6). Future work could focus on finding a model that would best generalise across various sarcasm domains. This could include better tuning, other BERT-based architectures and other transformer models. Such architectures can be the one developed by Potamias et al. (2019), RoBERTa (Liu et al., 2019), XLNET (Yang et al., 2019) or BERT-Uncased (Devlin et al., 2018).

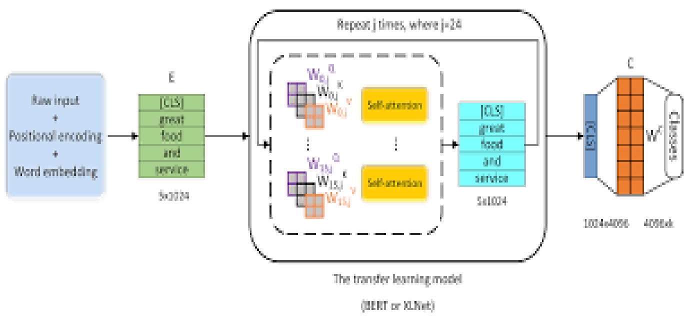


Figure 9: Transformer Classification Model (Tao and Fang, 2020).

References

2017. A large self-annotated corpus for sarcasm.

Kiran Baktha and BK Tripathy. 2017. Investigation of recurrent neural networks in the field of sentiment analysis. In *2017 International Conference on Communication and Signal Processing (ICCSP)*, pages 2047–2050. IEEE.

Clint Burfoot and Timothy Baldwin. 2009. Automatic satire detection: Are you having a laugh? In *Proceedings of the ACL-IJCNLP 2009 conference short papers*, pages 161–164.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

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

Aditya Joshi, Pushpak Bhattacharyya, Mark Carman, Jaya Saraswati, and Rajita Shukla. 2016. How do cultural differences impact the quality of sarcasm annotation?: A case study of Indian annotators and American text. In *Proceedings of the 10th SIGHUM Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*, pages 95–99, Berlin, Germany. Association for Computational Linguistics.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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

Baoli Li and Liping Han. 2013. Distance weighted cosine similarity measure for text classification. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 611–618. Springer.

Peng Liu, Wei Chen, Gaoyan Ou, Tengjiao Wang, Dongqing Yang, and Kai Lei. 2014. Sarcasm detection in social media based on imbalanced classification. In *Web-Age Information Management*, pages 459–471, Cham. Springer International Publishing.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Sherin Mary. 2019. *Explainable Artificial Intelligence Applications in NLP, Biomedical, and Malware Classification: A Literature Review*, pages 1269–1292.

Quinn McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space.

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

Soujanya Poria, Erik Cambria, Devamanyu Hazarika, and Prateek Vij. 2016. A deeper look into sarcastic tweets using deep convolutional neural networks.

Saurabh Porwal, Gaurav Ostwal, Anagha Phadtare, Mohini Pandey, and Manisha V Marathe. 2018. Sarcasm detection using recurrent neural network. In *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 746–748. IEEE.

Rolandos Alexandros Potamias, Georgios Siolas, and Andreas Georgios Stafylopatis. 2019. A transformer-based approach to irony and sarcasm detection.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144.

Cícero dos Santos and Maíra Gatti. 2014. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, Dublin, Ireland. Dublin City University and Association for Computational Linguistics.

Lohita Sivaprakasam and Aarthe Jayaprakash. 2017. A study on sarcasm detection algorithms.

Jie Tao and Xing Fang. 2020. Toward multi-label sentiment analysis: a transfer learning based approach. *Journal of Big Data*, 7(1):1.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Rikiya Yamashita, Mizuho Nishio, Richard Do, and Kaori Togashi. 2018. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237.

Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. 2017. Comparative study of cnn and rnn for natural language processing.

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

A Appendices

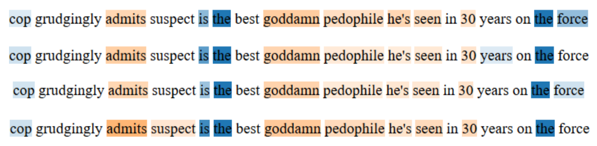
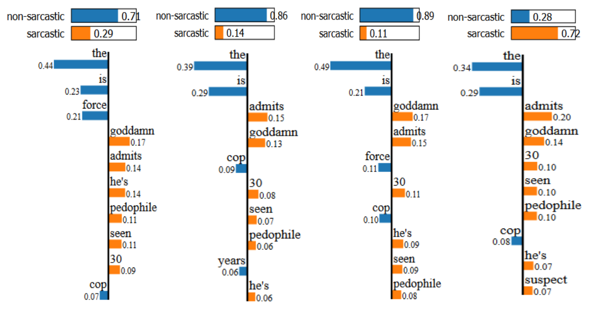


Figure 10: LIME Example 3



(a) RNN (b) LSTM (c) GRU (d) CNN

Figure 11: LIME Example 3

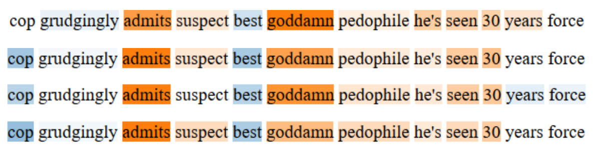
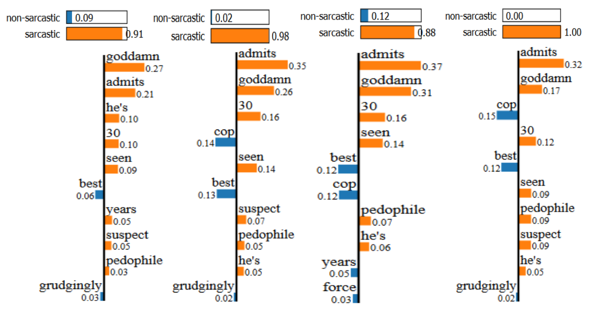


Figure 12: LIME Example 3 - No Stop Words



(a) RNN (b) LSTM (c) GRU (d) CNN

Figure 13: LIME Example 3 - No Stop Words

Dataset	Loss	Accuracy	F1-Score
News Headlines	0.2178	90.86%	0.9001
Reddit	2.0868	50.37%	0.1060

Table 6: BERT Results on News Headlines and Reddit datasets

Models	Labels	NoCheck			CheckSame			CheckCorrect		
			Similar	Average		Similar	Average		Similar	Average
RNN LSTM	Mixed	**500**	403	0.891	**455**	375	0.898	**407**	345	0.904
RNN LSTM	Sarcastic	**500**	327	0.862	**446**	310	0.873	**398**	277	0.874
RNN LSTM	Non-Sarcastic	**500**	433	0.911	**461**	407	0.916	**413**	373	0.921
RNN GRU	Mixed	**500**	399	0.897	**465**	375	0.899	**407**	335	0.904
RNN GRU	Sarcastic	**500**	341	0.868	**452**	327	0.879	**396**	288	0.884
RNN GRU	Non-Sarcastic	**500**	422	0.910	**457**	394	0.914	**412**	362	0.917
RNN CNN	Mixed	**500**	320	0.858	**446**	301	0.867	**401**	277	0.873
RNN CNN	Sarcastic	**500**	252	0.810	**431**	240	0.828	**385**	209	0.826
RNN CNN	Non-Sarcastic	**500**	385	0.888	**447**	353	0.897	**414**	338	0.902
LSTM GRU	Mixed	**500**	476	0.946	**470**	451	0.949	**416**	404	0.953
LSTM GRU	Sarcastic	**500**	468	0.938	**458**	435	0.944	**399**	385	0.948
LSTM GRU	Non-Sarcastic	**500**	480	0.950	**478**	462	0.951	**430**	417	0.953
LSTM CNN	Mixed	**500**	423	0.913	**465**	405	0.920	**417**	368	0.924
LSTM CNN	Sarcastic	**500**	395	0.895	**457**	371	0.903	**398**	322	0.908
LSTM CNN	Non-Sarcastic	**500**	461	0.937	**472**	443	0.942	**434**	412	0.946
GRU CNN	Mixed	**500**	424	0.904	**463**	401	0.911	**411**	368	0.916
GRU CNN	Sarcastic	**500**	384	0.884	**447**	358	0.897	**388**	318	0.900
GRU CNN	Non-Sarcastic	**500**	437	0.918	**470**	423	0.922	**434**	395	0.926

Table 7: This table shows the cosine similarity of LIME explanations for a combination of two models. Originally for each experiment we use 500 instances from the test set, either Mixed (both sarcastic and not), Sarcastic-only and Non-Sarcastic-only. Then we filter the instances based on the predictions of the models. NoCheck means that we do not check what the models predicted. CheckSame means that out of the 500 instances we keep only those that the two models classified with the same label and CheckCorrect only those that the two models classified correctly. The number of instances we keep after filtering for each experiment are indicated in bold in the table.