Final Project Report [CMPUT 651]

Mohammad Karimi Abdolmaleki
University of Alberta / Edmonton, AB
karimiab@ualberta.ca

Yashar Talebirad
University of Alberta / Edmonton, AB
talebira@ualberta.ca

Instructor: **Dr. Lili Mou**

Abstract

In this project, we demonstrate the problem of sarcasm detection and explain how we used different approaches from machine learning, deep learning, reinforcement learning, and knowledge graphs to tackle the problem. Moreover, we not only utilized state-of-the-art algorithms in NLP[1] but also tried to improve and expand the classification task into a more complicated task which results in increasing the performance and interpretability of the work simultaneously.

1 Introduction

Sarcasm is an ironic or satirical remark tempered by humor by representing something positive with negative intent. Nowadays, with the ever-growing rate of website construction and easy access to the internet and smartphones, using sarcasm or irony as a news headline, tweet, Facebook status, Reddit comment, or other forums is becoming usual. Consequently, understanding the presence of sarcasm in a sentence would be a challenge in the field that needs to be addressed with the state-of-the-art algorithms in deep learning and NLP techniques.
Moreover, in human-machine interaction and review mining, understanding the sentiment of a sentence is crucial since an appropriate detection of sentiments from a text will lead to correct decision-making. Furthermore, one can observe that one of the essential uses of sentiment analysis is websites' movie reviews and chatbot programs. Nowadays, most applications have some kind of assistant or a bot to help their customers in several ways. In order to be beneficial, these chatbots must maintain a human-like understanding of the user sentiments; otherwise, the output cannot be reliable for a user. This is also the case in movie review analysis; One can only predict the accurate intuition of a review if their sentiment analysis module has the ability to detect sarcasm. Detecting sarcasm is also crucial for social media analysis to differentiate between the two opposite polarities that a statement can convey. Considering the points mentioned above, the importance of sarcasm detection is indisputable; this is why there is still much research on sarcasm detection each year published on various journals and conferences.

[1] Natural Language Processing

2 Literature Review

Sarcasm detection is most commonly formulated as a classification task. As (Joshi et al., 2017) explains, sarcasm detection approaches are broadly classified into three types: rule-based, statistical-based, and deep learning-based. We will elaborate on some of the work done in each category.

2.1 Rule-based

Rule-based methods are approaches that identify sarcasm through specific evidence. For instance, in (Maynard and Greenwood, 2014), sentiment analysis is done on the hashtags of a tweet, and if the sentiment of the hashtags do not correspond with the rest of the tweet's sentiment, it is determined as sarcastic.
Another example is the detection of a positive verb and a negative phrase in a sentence. This is done in (Riloff et al., 2013) using an iterative algorithm. Both of the mentioned approaches use methods that are outclassed by the current deep learning and machine learning techniques. Although the idea might seem attractive, in our knowledge, most of the rules used by the rule-based methods can be captured and learned by state-of-the-art algorithms (e.g., deep learning) automatically.

2.2 Statistical-based

Statistical-based methods mostly consist of machine learning classification algorithms. In (Forslid

and Wikén, 2015), (Parde and Nielsen, 2018), and (Sarsam et al., 2020), a selection of machine learning techniques were used on the Amazon product reviews dataset and tweets from Twitter to classify text into sarcastic and non-sarcastic groups. The models implemented in these papers seemed interesting as a starting point. Thus, we implemented some of them (i.e., Support Vector Machine (SVM), Logistic Regression, and Naive Bayes) with our designed pre-processing methods.

Also in (Khatri and P, 2020), sacracsm detection is done on the Twitter dataset, using ML models and word embeddings. Bidirectional Encoder Representations from Transformers (BERT, first introduced in (Devlin et al., 2018)), and Global Vectors for Word Representation (GloVe, first introduced in (Pennington et al., 2014)) were used as word embeddings in this paper. Word embeddings are further explained in section 3.2. Applying word embeddings alongside the models seemed to be a good idea both in terms of performance and interpretability, which was why we used several types of word embeddings alongside the best performing models in our project.

Although these methods have yielded decent results, they fail to capture the context of a sentence and do not benefit from the sequential structure of the sentences and the relationship between words. Thus, we can expect them to perform worse than deep learning approaches. Despite that, some approaches have used Hidden Markov Models (HMM) alongside machine learning algorithms to make use of the sequential structure and get better results. For instance, (Wang et al., 2015) uses SVM-HMM to incorporate the sequence nature of output labels in a conversation. However, the model used in the paper relies heavily on certain contextual information of the tweets (i.e., history-based context, conversation-based context, and topic-based context) taken from the Twitter API, which were not present in our datasets. Because of this limitation, it was not possible to utilize this approach for our datasets.

2.3 Deep Learning-based

Deep learning-based approaches consist of state-of-the-art deep learning algorithms. These approaches have recently been proven useful in many NLP tasks. Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) such as LSTMs have all been extensively used in different classification tasks, such as sarcasm detection.

In (Felbo et al., 2017), Twitter data was used to create a deep learning model that captures the underlying emotion of a tweet and outputs an emoji (from a diverse selection of emojis) corresponding to that emotion. Since it is stated that the model can also capture sarcasm and slang, the work is closely related to the problem of detecting sarcasm alone. Thus, similar methods can be used to detect sarcasm and other emotions in a tweet. This can be regarded as a combination of deep learning-based and rule-based approaches. However, since the idea of extracting emojis does not expand the classification task in terms of complexity and accuracy, we decided not to use the technique of extracting emojis.

In (Salim et al., 2020), a Bidirectional LSTM alongside word embeddings are used. Also in (Ashok et al., 2020) and (Shangipour ataei et al., 2020), the same idea utilizing BERT, followed by a Convolutional Neural Network is used for sarcasm detection. The idea of using a bidirectional LSTM seemed to provide them with decent results, and thus, we decided to take benefit from this approach. We also utilized word embeddings in our implementation.

In (Sangwan et al., 2020), recurrent neural networks are used to exploit the interaction among the ”input modalities” for the prediction. The model feeds both text and image to deep neural networks in order to classify something like sarcasm. It is stated that using visual modalities is very beneficial for improving the performance of the model. However, it’s worth mentioning that many NLP datasets do not have images besides the text data, and even if they do, they rarely correlate with the sentiment of the text. Moreover, our goal is to focus our research on the NLP side and expand the mathematical aspect of the project. Thus, the aforementioned idea is not a good match for our task.

Furthermore, in (Petrželková et al., 2020), the authors constructed the meta-features based on knowledge graphs. They utilized Microsoft Concept Graph for obtaining the extra semantic information from the text data. Then, several document classification tasks were applied on eight different datasets using the knowledge graph additional features, which resulted in improved accuracy and interpretability of the model. In such a setting, it was shown that approaches capable of using semantic context might outperform the naïve learn-

ing methods. Adding background knowledge into a learning process may increase the accuracy and interpretability of the model, which can be done using knowledge graphs. To the best of our knowledge, sarcasm detection has never been done by utilizing knowledge graphs. As a result, we implemented sentence segmentation, entity relation, and extraction with the help of knowledge graphs.

2.4 Reinforcement Learning-based

Reinforcement learning algorithms are not only rich in terms of mathematical aspects but also, if formulated and applied correctly, can have several beneficial impacts. In (Minaee et al., 2021) which was published recently, the authors reviewed more than 150 deep learning-based models for text classification. In the RL part of the review, it is stated that (Shen et al., 2018) used a hard attention model to select a subset of critical word tokens of an input sentence. They considered the hard attention model as an agent that takes actions of whether to select a token or not. After going through the entire text sequence, it receives a classification loss, which can be used as the reward for training the agent. Moreover, in (Wiering et al., 2011), the mapping formulation from supervised setting into RL is based on the input data augmentation and defining agents for each class target. The augmented data is state vector that the agent has to interact with it in two way, either by copying the elements of the first bucket which is the original data in the second bucket or by removing the copy of the original data in the third bucket. The key idea is that since the agents have interacted with similar state vectors, there is no need for interaction on the data never seen during the test phase. Finally, the class agent who has the highest value for the initial state outputs the presented class.
In both of the described approaches, RL is used as an innovative and intuitive sub-module of the classification task. We will also try to take benefit from the mentioned techniques in a way to address the problem of sarcasm detection.

3 Data

3.1 Datasets

There are several datasets in the task of Sarcasm Detection. Among all of them, there are two datasets that we're interested in, which are called "News Headlines Dataset For Sarcasm Detection", which has more than 28600 records, and "Sarcasm on Reddit" which has more than 1.3 million records. The first dataset has three features, the article link, the news headline and, the target label. It is evident that extracting features and pre-processing steps have to be done on both of the datasets. The second dataset has ten features, including the user's comment, parent comment, date, subreddit, label, etc. This is a huge dataset that also needs to be prepared before applying machine learning and deep learning models. In both datasets, the label is a binary value that can be either zero or one, depending on the absence or presence of sarcasm in the sentence, respectively. Also, for detecting irony in a sentence, the essential feature is the text itself which has all the necessary aspects of detecting the sarcasm.
We have also applied several exploratory data analysis (EDA) techniques to investigate both News Headlines and Reddit Comments dataset. For instance, In both the News Headline and Reddit Comment datasets, the distribution of the target variable is balanced. This can be seen in figures 1 and 2.

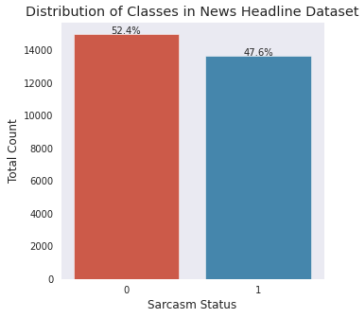

Figure 1: Distribution of Classes in News Headline Dataset

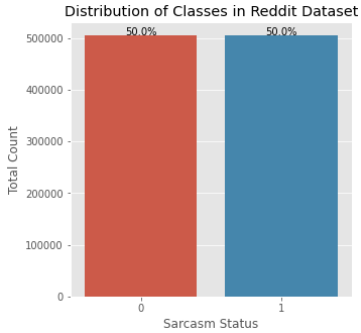

Figure 2: Distribution of Classes in Reddit Comment Dataset

Moreover, we have done several exploratory data analysis tasks in order to understand the data and

have an intuition for selecting our potential approaches. The experimental logs feature some of them.

3.2 Data pre-processing

In most cases, there is a direct relation between the quality of the input and output data. Usually, the raw data is not complete or has some useless information, which only adds to the problem's ambiguity. Thus, applying pre-processing techniques makes the data suitable for analysis. The general approach for the text data pre-processing is taken from Mayo which is elaborated below:

- Handling Null Values: In each dataset, it is probable to see null values in the data. The presence of the null data leads toward an inaccurate estimation of the data. Thus, the null samples should be either removed or replaced by the mean of the particular feature.

- Tokenization: The task of splitting the sentences into different elements. Since computers don't have any understanding of the words, we need to transform each word into a number. By doing this, the main challenge is preserving the relationship between the words and their corresponding numbers. This challenge is solved by a concept named embedding. Word embedding is one of the most popular representations of document vocabulary. It is capable of capturing the context of a word in a document, semantic and syntactic similarity, relation with other words, etc. In NLP, word embedding is a term used for the representation of words, typically in the form of a real-valued vector that encodes the meaning of the word such that the words are closer in the vector space is expected to be similar in meaning. Since the words are represented as vectors, the objective is to have words with similar contexts occupy close spatial positions. For example, if we have words as one-hot vectors, then the cosine of the angle between such vectors should be close to one. (i.e., the angle should be close to zero) which is defined as:

$$Sim(A, B) = \cos\theta = \frac{A.B}{|A||B|}$$

  Word2Vec is one of the most popular techniques to learn word embeddings using a shallow neural network. It was developed by Tomas Mikolov in 2013 at Google. It can be achieved using a neural network with CBOW or SkipGram. Both of these methods have their own advantages and disadvantages. According to Mikolov, SkipGram works well with a small amount of data and is found to represent rare words well. On the other hand, CBOW is faster and has better representations for more frequent words. Tokenization usually comes with punctuation removal, which helps to remove unnecessary notations of text for sarcasm detection. Moreover, it is also a great idea to lower case the text data unless the whole word is written in capital letters.

- Stopword Removal: A similar idea to the above-mentioned punctuation removal. The basic idea is to remove the information which is not necessary for the task. There are many stop words in English (i.e., the, a, an, etc.) that add no useful information to the text.

- Stemming: The process of converting the word into its root form. This will usually result in easier processing.

- Normalization: The process of normalizing text. (i.e., converting text into its standard format.) Since both of the datasets that we're using consist of sarcastic sentences, the probability of having an unnormalized word like "Hellllloooo" is high. Normalization techniques transform the word mentioned above into its standard format, "Hello."

Since we are using two different datasets, we need to use a combination of the aforementioned methods for pre-processing as well. We even found out that different types of pre-processing will result in different outcomes between several models. Thus, we had to try the stages mentioned above to see their effects on our models. For the News Headline dataset, we concluded that two different pre-processing steps could be done. For machine learning models, we first clean the data, then we use a tf-idf or count vectorizer to obtain the vectors. This results in a huge sparse matrix, which is suitable for machine learning models that mostly use linear algebra methods to obtain the result but is not feasible to be fed as input to Keras and similar deep learning models due to its extensive nature. Instead, for deep learning models, the default tokenizer from Keras is used with a constraint on

the maximum number of words. The processing consists of tokenizing and then padding the data. This maximum number of words was fine-tuned, and 10000 was proved to be a sufficient number in some of our tests. We also learned that by not cleaning the text (removing stopwords, punctuations, etc.), the model would be prone to overfit and not generalize well on the data never seen before. For the Reddit dataset, we also used a combination of pre-processing techniques mentioned above with semantic-based word embeddings to prepare the data for applying the models on them.

4 Analysis / Methodology

In order to find out which model works best for our problem, we not only studied the papers mentioned above but also took a look into many of the codes and algorithms submitted both in the codes section of the corresponding Kaggle challenges and GitHub repositories, for the News Headline dataset and the Reddit dataset, respectively. By understanding the behavior of each model and applying some of the previous work on our data as a starting point, we came up with several machine-learning models for the two datasets on which we used a number of different methods for pre-processing and hyper-parameter tuning.

In most of the aforementioned related work, it can be seen that neural network algorithms are crucial for achieving an acceptable amount of any evaluation metric in detecting sarcasm. In this project, we first tried to experiment with some of the approaches and found a good combination that gave us decent results. We also learned that state-of-the-art algorithms and word embeddings (i.e., RNN, LSTM, BERT, GloVe) have more ability to classify a sentence as sarcastic in the correct way due to their complex calculation and high computation power. Long Short-Term Memory (LSTM) networks are a particular type of recurrent neural network capable of learning order dependencies in sequence prediction problems. In fact, remembering information for a long period is their default behavior provided by the concept of cell states. This behavior helped us have more true positives and true negatives on the test data compared to the other models.

Furthermore, we tried to extract the entities and the relations to build a knowledge graph. Knowledge graph representation can add background knowledge to the text, which can be interpreted as a semantic feature. This can also become beneficial in situations of insufficient data available (News Headline dataset).

Finally, to the best of our knowledge, the problem of sarcasm detection has never been addressed with the help of RL. This tempted us to explore some of the previous work in classification with the help of RL and try to formulate the problem into the RL setting.

We will now explain each of the aforementioned approaches in the following subsections.

4.1 Machine Learning Models

There are various machine learning algorithms that can handle the classification task. At first, we tried two basic algorithms for both datasets: Logistic Regression and Naive Bayes. Lastly, we tried Support Vector Machines. We will elaborate on these methods in the subsequent paragraphs.

4.1.1 Naive Bayes

Naive Bayes algorithms are a family of algorithms based on the Bayes theorem. The naive assumption in this algorithm is that every feature is independent of the others in order to predict the category of a given sample. In other words, the Naive Bayes classifier refers to the conditional independence of each of the features in the model, and the Multinomial Naive Bayes is a specific instance of a Naive Bayes classifier that uses a multinomial distribution for each of the features. Formally, if we want to calculate the probability of observing features f_1 through f_n, given some class c, under the Naive Bayes assumption, the following holds:

$$P(f_1, ..., f_n|c) = \prod_{i=1}^{n} P(f_i|c)$$

Which means that if we want to classify a new example, we have to calculate the probability of each category using the Bayes theorem:

$$P(c|f_1, ..., f_n) \propto P(c)P(f_1|c)...P(f_n|c)$$

And then the algorithm outputs the category with highest probability:

$$\hat{c} = \arg\max_{c} P(c) \prod_{i=1}^{n} P(f_i \mid c)$$

In this case, we used the multinomial Naive Bayes. The term "Multinomial" simply lets us know that each $p(f_i|c)$ is a multinomial distribution.

Finally, to calculate the probabilities, the Naive Bayes algorithm should count the number of words in each class and count the number of occurrences in each specific class.

$$\frac{count(w,c)}{count(c)} = \frac{\text{counts of w in class c}}{\text{counts of words in class c}}.$$

Note that this estimation of $P(w|c)$ could cause some problems since it would yield a probability of zero for documents with unknown words. A common way of solving this problem is to use Laplace smoothing. Define

$$P(w|c) = \frac{count(w,c)+\alpha}{count(c)+K\alpha}$$

Where k denotes the number of different values each feature can take, and α is defined as a smoothing parameter. For Naive Bayes, we tried to see the effect of the smoothing parameter (α) on the training, but we found out that increasing α will cause a decrease in the accuracy.

4.1.2 Logistic Regression

Logistic Regression is a statistical model used to determine if an independent variable affects a binary dependent variable. This means that given input, there are only two potential outcomes. The basic idea is similar to linear regression, which was to fit a straight line through the data, while in this case, we try to fit an s-shaped curve through the data. The logistic regression is basically the sigmoid function applied to the weighted sum. The sigmoid function is shown in figure 3.

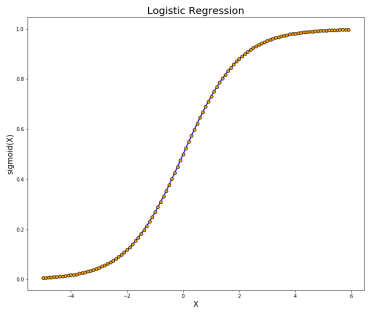

Figure 3: Sigmoid Function

We assume a linear relationship between the predictor variables and the log-odds (also called logit) of the event that $y = 1$. To make use of this idea in learning, we first need to consider a linear model function parameterized by θ.

$$h_\theta(X) = \frac{1}{1+e^{-\theta^T X}} = \Pr(y=1 \mid X;\theta)$$

Then, we calculate the likelihood function:

$$L(\theta \mid y;x) = \prod_i \Pr(y_i \mid x_i;\theta)$$

By assuming that all observations in the sample are independently Bernoulli distributed, we have:

$$L(\theta \mid y;x) = \prod_i h_\theta(x_i)^{y_i}(1-h_\theta(x_i))^{(1-y_i)}$$

And the objective will be to find the "best" θ. Afterward, the gradient descent is used on the derivative of the log-likelihood in order to find the optimal parameters.

4.1.3 SVM

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. The algorithm performs very well with a limited amount of data to analyze. By giving data points to an SVM, it tries to find the "best" hyperplane that separates the data by their target variable. The separating hyperplane is called the "decision boundary." Between all hyperplanes that separate the data, the "best" one is considered to be the one that has the maximum margin to the nearest data points with opposite classes (which are called support vectors). Mathematically speaking, the problem of finding the best hyperplane turns to:

$$(\vec{w}^*, b^*) = \arg\max_{\vec{w},b} \min_i \frac{\left|\vec{w}^T\vec{x}_i + b\right|}{\|\vec{w}\|}$$

In which we should have $\forall i,\ y_i(\vec{w}^T\vec{x}_i + b) \geq 0$. If the data cannot be separated by a line, the SVM will classify the data by mapping the feature space into a higher dimension, in which the mapped data will be linearly separable, and the classification can be done, also known as kernels.

4.2 Deep Learning Models

Due to some insufficiency in terms of accuracy and model performance obtained with the ML models and the simplicity of them, we decided to implement some of the neural-based models. In fact, since we observed that even in 2021, there are still ongoing research and publications in the field of sarcasm detection, using deep learning[2], we've decided to apply DL models to both of the datasets. We will explain the methodology, the intuition behind each of these approaches, and the experimental results that we obtained with these models in the following sections.

[2]https://www.aclweb.org/anthology/venues/acl/

4.2.1 Sequential

A sequential model consists of a stack of layers, with each layer having exactly one input and one output tensor. After tokenizing each data point, the network begins with taking the vectors as input to its embedding layer and turns them into dense vectors of fixed size. Afterward, the network proceeds as the architecture in figure 4 shows.

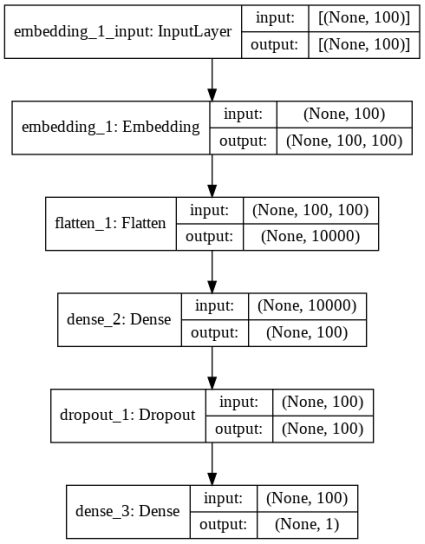

Figure 4: Architecture of the sequential model

This network was proved to be enough for the news headline dataset and provided us with a decent accuracy, which is further explained later in the experimental results section. For the Reddit dataset, a similar sequential model was tested, but the accuracy was not as good as the LSTM model, which will be explained in the following subsection.

4.2.2 Bidirectional LSTM

LSTM stands for Long Short Term Memory, which is a type of recurrent neural network. The main advantage of this algorithm is taking benefit from the concept of cell state, which can be thought of as an information pipeline or a memory slot that can store more data compared to the vanilla RNN models.

In each LSTM cell, there are three essential valves that are crucial to the model. Forget valve, which decides what part of the sentence should be forgotten from the previous cell state and what parts should remain; Memory valve, which decides how much new memory should influence the old memory; Output valve, which generates an output of this particular LSTM cell and is controlled by new memory and previous output. This valve controls how much new memory should output to the next LSTM unit. The architecture of the Bidirectional-LSTM model used in this project is shown in figures 5 and 6.

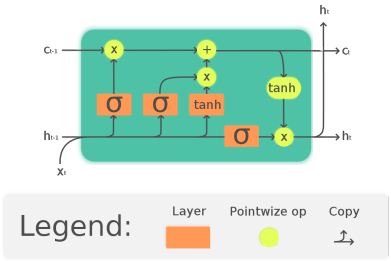

Figure 5: Structure of the LSTM cell

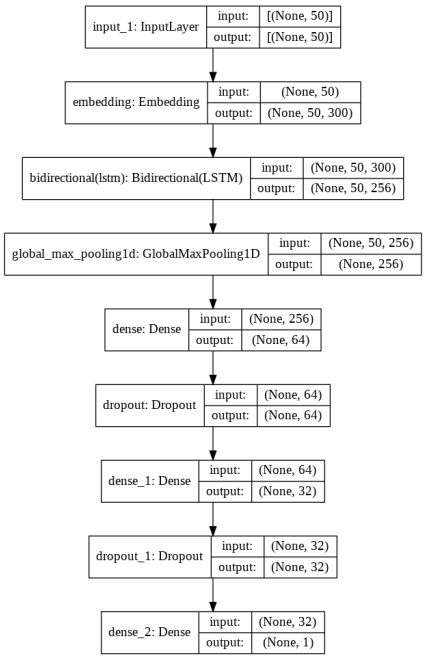

Figure 6: Architecture of the LSTM model

The input layer indicates the text sentences that should be passed to the embedding layer. In the embedding layer, each word is mapped with its corresponding embedding vector. Also, the "FastText crawl 300d 2M" from Facebook was used as the embedding, which includes more than 2 million word vectors that are trained on Common Crawl. The LSTM layer is the next with 128 LSTM cells. In this layer, a Bidirectional LSTM was used to give the model a double-sided context, which usually results in a better performance. The max-pooling layer is to calculate the maximum value for each

patch of the feature map; this results in a summarized version of the features detected in the input. The feed-forward dense layer is to classify the features captured by the LSTM layer. In the dropout layer, a percentage of the neurons will be randomly turned off since the network should not depend on specific neurons to make predictions. This will cause the training not to overfit the data and makes the training robust. The second dense and dropout layers are the same. The output layer is the final layer which is to get the output prediction from the model.

4.2.3 Bidirectional LSTM-GRU with Word2Vec

In this experiment, we did a combination of LSTM with Bidirectional Gated Recurrent Unit (GRU) to overcome the problem of vanishing gradients. The GRU controls the flow of information like the LSTM unit, but without having to use a memory unit. The architecture of the used model is shown in figure 7.

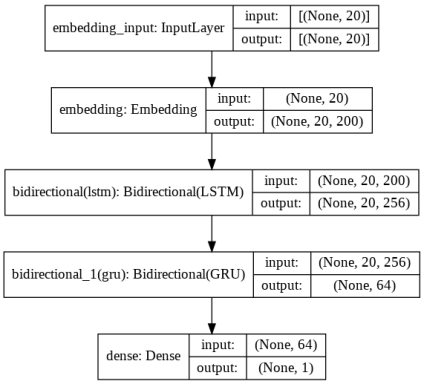

Figure 7: Architecture of the BiLSTM-GRU model

4.2.4 Global Vectors for Word Representation (GloVe)

GloVe method is built on an important idea. You can derive semantic relationships between words from the co-occurrence matrix. It is actually an unsupervised learning algorithm for obtaining vector representations of words (Pennington et al., 2014). Given a corpus having V words, the co-occurrence matrix X will be a $V \times V$ matrix, where the i_{th} row and j_{th} column of X, X_{ij} denotes how many times word i has co-occurred with word j. An example co-occurrence matrix might look as shown in in figure 8.

	the	cat	sat	on	mat
the	0	1	0	1	1
cat	1	0	1	0	0
sat	0	1	0	1	0
on	1	0	1	0	0
mat	1	0	0	0	0

Figure 8: Co-occurrence matrix for a given sentence

The algorithm maps words into a meaningful space where the distance between words is related to semantic similarity. In figure 9 it can be seen that although "barrack obama" and "donald trump" are different in terms of characters, but the semantic similarity between them is high, according to the figure, indicating that both of them were president of the U.S.

Similarity with GloVe

	phrase	score
0	barrack obama	0.956801
1	barrack h. obama	0.944671
2	barrack hussein obama	0.937000
3	michelle obama	0.905201
4	donald trump	0.729601
5	melania trump	0.614963

Figure 9: GloVe similarity

For achieving a noticeable result with GloVe word embedding, we have implemented an architecture as shown in figure 10 with choosing the Bidirectional LSTM as the primary model and GloVe as the embedding layer. A dense layer is also applied after the LSTM layer.

4.2.5 BERT and Attention

BERT stands for Bidirectional Encoder Representations from Transformers, introduced in (Devlin et al., 2019). While the previous works on NLP look at a text sequence either from left to right or combined left-to-right and right-to-left training, BERT applies the bidirectional (or, in better words, non-directional) training of the Transformer, a pop-

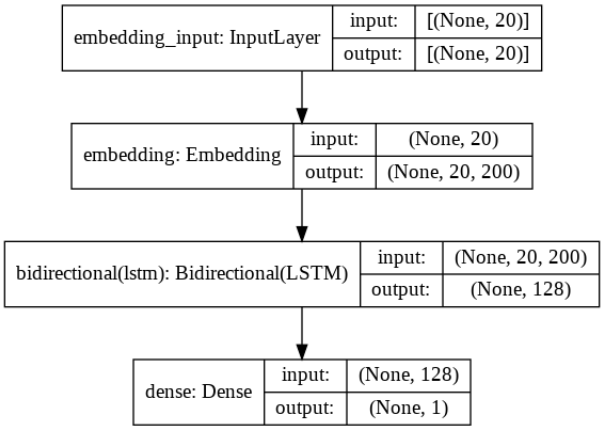

Figure 10: Architecture of the BiLSTM GloVe model

ular attention model, to language modeling. In BERT, an attention mechanism lets each token from the input sequence (e.g., sentences made of word or subwords tokens) focus on any other token. The concept of attention was first introduced in (Yang et al., 2016), and thus it is considered relatively new. With the methods used in the tokenization of the sentences in Machine Learning models, we concluded that some words might contribute more towards determining the existence of sarcasm in a sentence (see figures 15 and 16). However, it was evident that by doing this, we are not using the sequential structure of the sentence, which is indeed very helpful, as shown in the deep learning models (especially LSTM). By using attention, we are able to use both of these features. As it is told by the author of the paper, "Not all words contribute equally to the representation of the sentence meaning. Hence, we introduce an attention mechanism to extract such words that are important to the meaning of the sentence and aggregate the representation of those informative words to form a sentence vector".

In short, as this blog has explained simply, we create scores for every word in the text, which are the attention similarity scores for each word.

Back to BERT, as explained in here, the transformer encoder reads the entire sequence of words at once. This will allow the model to learn the context of a word based on all of its surroundings. Thus, a language model which is bi-directionally trained can understand the language context and flow much better than single-direction language models. Since sarcasm is a sentiment that requires a deep understanding of context to be identified, we believe that it would be a good match to use BERT in the task of sarcasm detection. One of the training strategies of BERT is called Masked LM (MLM). In this strategy, at first, $k\%$ of the words in each sequence is replaced with a [MASK] token. The model then attempts to predict the original value of the masked words based on the context provided by the other non-masked words. In this project, we took benefit from BERT alongside a neural network.

4.3 Knowledge Graphs

So far, we have implemented several machine learning and deep learning algorithms capable of detecting sarcasm with an acceptable accuracy. To the best of our knowledge, sarcasm detection has never been done by utilizing knowledge graphs or reinforcement learning algorithms. In this section, we will try our best to use both of these concepts in order to expand the classification task.

Knowledge graphs (KG) represent a collection of interlinked descriptions of entities. It is a way of storing data that resulted from an information extraction task. The basic implementation of a knowledge graph uses a three items data type (a subject, a predicate, and an object) which is called a triplet. As shown in figure 11, nodes A and B are connected with an edge which is the relationship between two nodes.

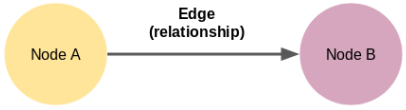

Figure 11: A simple knowledge graph

By utilizing the power of knowledge graph, we will be able to represent the data in a better way that is more suitable to the neural network model. To build our knowledge graph from the text, we need to apply several NLP techniques. (i.e., sentence segmentation, dependency parsing, extracting entities, etc.) It is also worth mentioning that for constructing knowledge graphs, we used some tutorials on the web and kaggle website.

4.3.1 Sentence Segmentation

The first step in constructing the knowledge graph is to split the text into chunks. Since we have both the headlines and the Reddit dataset as a data frame, chunking would be simple.

4.3.2 Entities Extraction

We are using the spaCy module for doing this task. Consider an arbitrary sentence from the NewsHeadline dataset: ”god getting strong urge to bring back dinosaurs,” the extraction of a single word entity from a sentence is not difficult. It is indeed feasible with the help of part of speech (POS) tags. However, when an entity spans across multiple words, then we need the dependency tree of the sentence. So, for constructing the knowledge graph, we need to have nodes and relationships, the nodes would be the entities that are present in the dataset, and the edges would be the relationships that would connect these nodes to each other. As a result, the main idea is to go through the sentence and extract the subject and the object and their relationship, but as mentioned above, an entity can span across multiple words, and the dependency parser tag only the individual words as subjects or objects. The entities extraction module's aim is to return the entities (subject and object) of the sentence. The module result for the sentence 'The paper has more than 2000 citations.' is 'paper 2000 citation.'

4.3.3 Relation Extraction

The entity extraction yields the crucial elements of the sentence. However, there is still another step to construct the knowledge graph; we need edges to connect nodes (entities) to each other. In our case, the predicate is the main verb in the sentence. For this problem, we used spaCy's rule-based matching to catch the main verb. For example, the module outputs 'completed' as the main verb of the 'Mohammad completed his assignment' sentence.

4.3.4 Constructing Knowledge Graph

Now that we obtained entities and relationships of our data, the construction of the knowledge graph is feasible. We are using networkx module to represent the knowledge graph of our dataset. Since the knowledge graph representation of the entire dataset is not going to be intuitive, we have created an example of the 'read' relation, which is among the relations extracted from the data.

As shown in figure 12, some of the entities were extracted correctly, and some were not. For example, ”college graduate reads book” is a correct example represented in the graph. The reason is that the data is raw, and several techniques from pre-processing section should be applied to the data in order to make it ready for constructing the knowledge graph.

Figure 12: A knowledge graph with 'read' as relation

4.4 Reinforcement Learning

Reinforcement learning (RL) is a method of training an agent to perform discrete actions according to a policy, which is trained to maximize a reward (Sutton and Barto, 2018). However, applying a reinforcement learning algorithm to a classification task can be a challenge.

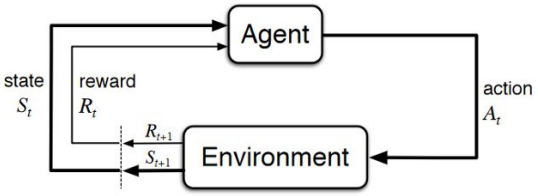

Figure 13: The agent-environment interaction in a MDP

RL algorithms are different from supervised learning algorithms described above. In a supervised task, an input is mapped to a desired output by utilizing a dataset of labeled training instances. The key difference is that the RL agent is never told the optimal action, but it receives an evaluation signal indicating the quality of the chosen action. Consequently, instead of propagating the input data through the model, an RL agent could be used to interact with the input data by performing

actions and manipulating the input representation. In this way, our belief is that the agent can find some hidden aspects of the input data that were never accessible through original input vectors.

4.4.1 Formulation to RL Setting

The RL agent needs to take action for being able to maximize the cumulative future rewards. Thus, action space and reward function need to be defined. Our mapping from supervised setting into RL is based on (Wiering et al., 2011) idea with some differences. The agent can perform actions, setting memory cells to particular values. These cells can encode additional information and can be used by the agent beside the original input vector for selecting actions. The agent has a working memory which is made of two memory cell slots. As a result, the agent has three slots. The first one is the input vector, the second one is the 'select' slot, and the third one is called the "ignore" slot. The second slot is initially empty, and the third slot initially contains a copy of the input data. The agent can modify its slots in two ways. It can copy the input data in the second slot so that it will have three copies of the input or it can delete the third slot and making the input data unique. An agent representing a particular class should copy the inputs, and agent representing the other class of the received sample should clear the memory slots. The reward function for this problem is class label independent, and it is combined with an agent that either tries to maximize or minimize rewards. The agent can receive a reward for setting particular memory cells and punishment for setting other memory cells. Consequently, this reward function can be used for learning a value function that can be used for selecting the actions in an optimal way. The mentioned idea can be done by using the actor-critic learning automation (ACLA), which basically is consists of actor and critic networks. The actor network chooses an action at each time step, and the critic network evaluates the quality of a given input. As the critic network learns which states are better or worse, the actor uses this information to avoid bad states. Thus, as the agent interacts with training samples, it will learn positive values for patterns belonging to its class and negative values for others. The learning process is done when the agent acts on the slots and receives a reward for setting slot cells. After the learning phase, the agent can use the value of the initial state for evaluating unseen records since the initial value is a summary of all of the processes and can be thought of as a weight matrix.

In the supervised setting, x^i is an input vector of size m and y^i the label belonging to this particular input. The algorithm is provided with a dataset $D = \{(x^1, y^1), ..., (x^n, y^n)\}$ which y^i can take 0 or 1 values. However, we still need to model the classification task into a sequential decision-making problem. The basic idea is to have one agent for every target variable and let them interact with the input data. As mentioned above, an agent receives a state vector, which consists of three slots. Each slot has the same length as the input vector. The first slot C_1 is the actual input x^i, so that the agent is always allowed to see the input. The second slot C_2 is initially set to zeros and can be set to the elements of the input vector by the agent. Finally, the third slot C_3 is initially set to a copy of the input vector and can be set to zero by the agent. So, the length of the new input vector is three times of the original input vector. Also, the agent has actions to set the cells of the C_2 and C_3 slots. For the C_2 slot, the agent can set the value of each cell to the corresponding input vector, and for the C_3 the actions are setting each of the cells to zero. The reward function is class-independent and is based on the number of zeros in the last two slots. The number of zeros can be denoted as $0 \leq z \leq 2m$, so the reward achieved after each action is $r_t = 1 - \frac{z}{m}$ which would be between -1 and 1. Despite the fact that the reward function is class independent, the agent with the same class as the training sample will select actions that maximize its achieved reward while the agent from another class will select actions that minimize the achieved rewards (Wiering et al., 2011). Below representation is an imaginary example of the augmented input data and the agent's interaction on it.

x_1	x_2	x_3	0	0	0	x_1	x_2	x_3

The agent performs action on the second cell of C_2 slot. $Reward = 0.33$ $(i.e., 1 - \frac{2}{3})$.

x_1	x_2	x_3	0	x_2	0	x_1	x_2	x_3

Also, the agent can remove cell from C_3, in that case, $Reward = 0$ $(i.e., 1 - \frac{3}{3})$.

x_1	x_2	x_3	0	x_2	0	x_1	0	x_3

As a result, an optimal agent would have three copies of the input sample in its slots, and the other class agent would only keep the data in its first slot and removes the sample from its third slot. The

problem specification and notation can be written as:

- S denotes the state space. For an input vector x^i with length m the state $s^i \in S$ has $3m$ elements.
- A denotes the action space. Since the agent can only modify two slots, there are $2m$ actions available in the action space. a_t denotes the chosen action by the agent at time step t.
- s_0 which is the initial state of an input vector x^i is equal to $(x^i, \vec{0}, x^i)$.
- With the descriptions mentioned above, the transition function T should be deterministic. It copies the previous state and then performs the action. In this case we have two different situations. First, when $0 \leq a_t < m$ the new state $S_{t+1} = O(s_t, a_t)$ which X is an operator that executes the effect of the action. In this case, the $(m + a_t)^{th}$ element of the C_2 slot would be set to a copy of the a_t^{th} element of the input vector. Second, if the action satisfies $m \leq a_t < 2m$, $S_{t+1} = O(s_t, a_t)$. In this case, the $(m + a_t)^{th}$ element of C_3 slot would be set to zero.
- Reward function is based on the number of zeros and is equal to $r_t = 1 - \frac{z}{m}$.
- A boolean variable is used in the learning phase that specifies whether the agent represents the same class as the instance. (i.e., whether to maximize or minimize the reward)
- The discount factor γ basically decides how much reinforcement learning agent care for rewards in the future versus those in the present. If this parameter is set to 0, the agent will be entirely myopic, only learning about actions that result in an immediate reward. If this parameter is set to 1, the agent will determine each of its actions based on the amount of all possible rewards.

4.4.2 Learning and Test Phases

In the learning phase, the class agent observes and interacts with the training samples, one at a time. In each iteration of the learning phase, an instance is given to both of the agents to perform a fixed number of actions on them to learn from the observed transition and obtained reward. After a sufficient number of epochs, the agents can be used for testing on the unseen instances. The class agent that obtains a higher value for the initial state can output its corresponding label as a result of the classification. To obtain a value function, we intend to use a sequential neural network as our function approximator. It is obvious that the learning phase for the RL setting would take much more time than a learning process of a normal neural network since the class agents need to compare the outputs of all actions to select the optimal action. ($2m$ actions)

4.4.3 Actor Critic Algorithm

During the training process, the aim is to learn large values for instances that their target variable corresponds to the correct class agent and low values for the agent with another class. Also, since we want to use the initial state vector and not to select actions anymore during the test phase, we need to utilize the actor-critic algorithm. It is also worth mentioning that we can not easily use Q-learning algorithms since the idea in this learning algorithm is based on taking actions, but in the proposed setting, it is impossible to take actions without knowing the true label.

The actor-critic learning algorithm uses an agent for each class i. It uses a state-value function V_i for selecting the actions. As mentioned above, for representing the value-function we use a sequential neural network. When an action is performed, the agent obtain information including (s_t, a_t, r_t, s_{t+1}). The agent has to compute the TD error (temporal difference) as below when the time step is less than the fixed limit of the number of trials. (i.e., horizon) (Wiering et al., 2011)

$$\delta_t = r_t + \gamma V_i(s_{t+1}) - V_i(s_t)$$

When the time step is equal to number of allowed trials for the agent in each epoch, the TD error can be calculated as follows:

$$\delta_t = r_t - V_i(s_t)$$

Thus, the TD update will effect the value function:

$$V_i(s_t) = V_i(s_t) + \alpha\delta_t$$

α denotes the learning rate of the critic. Having the δ_t, the target value for the actor of a specific action can be computed by:

$$G = \begin{cases} 1, & \text{if } \delta_t \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Which the G value is used as a target for learning the actions network. When the agent has to select actions for class $y = i$ during the training, it needs to maximize its obtained reward to learn high state values. Thus, the agent can utilize the softmax exploration policy. (Wiering et al., 2011)

$$P(a) = \frac{e^{P_i^a(s_t)}}{\sum_b e^{P_i^b(s_t)}})$$

Also, the agent has to learn values for the other class, in that case we can utilize the minimum of the above equation as the exploration policy.

$$P(a) = \frac{e^{-P_i^a(s_t)}}{\sum_b e^{-P_i^b(s_t)}})$$

As a result, the agent would try to obtain negative rewards for wrong classes which will be passed by TD learning to the value of initial state. Finally, we compute all values of $V_i(s_0)$ for both classes and agents related to these classes. At last, the test instance would be classified with label y_p to the agent with the highest state value:

$$y_p = \arg\max_i V_i(s_0)$$

5 Experimental Results

In this section, we have shared the results of the models we implemented and tested for both datasets. The plots and confusion matrices for under-performing models were omitted from the report and are instead presented in the experimental log, due to being sub-optimal. It's worth mentioning that the best-performing models for the Reddit dataset can be used in more contexts comparing to the models that are trained on the News Headline dataset, because news headlines are mostly short and should obey certain rules.

5.1 Machine Learning Models

5.1.1 Naive Bayes, Logistic Regression and SVM

For the News Headline dataset, the Naive Bayes and Logistic Regression methods provided us with acceptable results, with Naive Bayes being the superior one. Logistic Regression managed to achieve a 81.72% accuracy, while Naive Bayes achieved 86.99% accuracy. We also found out that increasing the smoothing value will decrease the accuracy.

Even though the previous two methods resulted in acceptable overall accuracy, between the machine learning models SVM model gave us a substantial improvement in accuracy of 93.41%, and its confusion matrix is shown in figure 14.

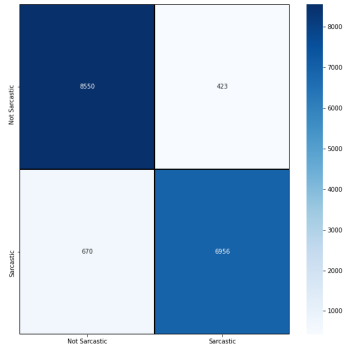

Figure 14: Confusion matrix of SVM on news dataset

5.1.2 Interpretation

One advantage of machine learning against deep learning is interpretability. Using libraries such as LIME, we are able to interpret how our models make a decision on certain instances. "The key intuition behind LIME is that it is much easier to approximate a black-box model by a simple model locally (in the neighborhood of the prediction we want to explain), as opposed to trying to approximate a model globally."[3] For instance, we tried to see how the Logistic Regression model works on a misclassified sentence and a correctly-classified sentence. The first sentence is: "the inside story of how congress sent the stock market tumbling", which is not sarcastic. The LIME interpretation on the logistic regression model is shown in figure 15.

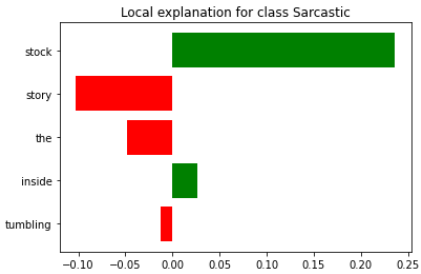

Figure 15: LIME interpretation of Logistic Regression on a misclassified sentence

[3] https://www.oreilly.com/content/introduction-to-local-interpretable-model-agnostic-explanations-lime/

We can see that the word ”stock” is attributed heavily towards the sarcastic class, which makes sense, considering that most headlines that have the word ”stock” in them are in the sarcastic class. However, this causes the model to make an incorrect prediction in this case. The second sentence is: ”jealous gps clearly wants man to back over wife”, which is sarcastic. The LIME interpretation on the logistic regression model is shown in figure 16.

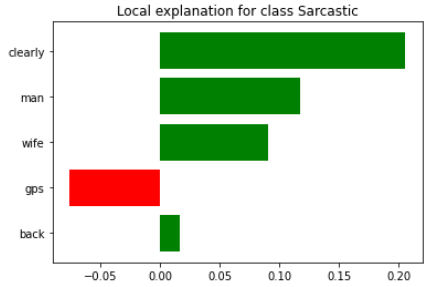

Figure 16: LIME interpretation of Logistic Regression on a correctly-classified sentence

We can see that the words ”man” and ”clearly” has made the classifier believe that this should be a sarcastic sentence, which is the case for this sentence. For the huge Reddit dataset, the accuracy achieved with Naive Bayes was 62%, and Logistic Regression provided us with a 72% accuracy.

5.1.3 Comparison

The next table compares the performance of different ML models that are used for different datasets.

Paper	Dataset	Model	Best Accuracy
(Forslid and Wikén, 2015)	Amazon	Naive Bayes	83%
		SVM	87%
(Sarsam et al., 2020)	Twitter	Naive Bayes	68%
		Logistic Regression	80%
		SVM	92%
(Khatri and P, 2020)	Twitter	Logistic Regression + GloVe	0.69 (F1)
(Parde and Nielsen, 2018)	Amazon	Naive Bayes	75%
	Twitter	Naive Bayes	53%
Our Project	Reddit	Naive Bayes	62%
		Logistic Regression	72%
	News Headline	Naive Bayes	87%
		Logistic Regression	82%
		SVM	94%

5.2 Deep Learning Models

5.2.1 Sequential Model

For the news headline dataset, the sequential model provided us with great results. The model was trained for 10 epochs and yielded 99.99% accuracy on the train data, and 92.93% accuracy on the test data. The training set and validation set accuracy plot is shown in figures 17.

For the Reddit dataset, the sequential model

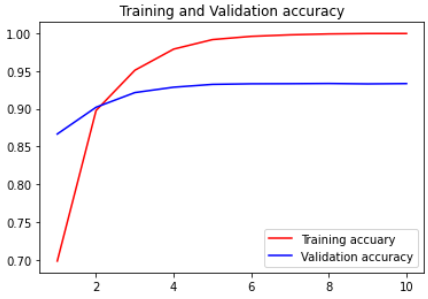

Figure 17: Training and validation accuracy for the sequential model on news dataset

achieved similar results to the Machine Learning models with a 67.21% accuracy which is still not satisfactory.

5.2.2 LSTM-GRU with Word2Vec

Using the combination of LSTM and GRU with Word2Vec embedding resulted in a good result for the news headline dataset. However, due to the model configuration we encountered the problem of overfitting. That is, the model was not generalizing well on the data that it has never seen before. However, a timely termination in the training process can save the model from overfitting. As shown figure 18, the model was capable of obtaining a decent accuracy on the training set. The model also achieved 80% accuracy on the test set.

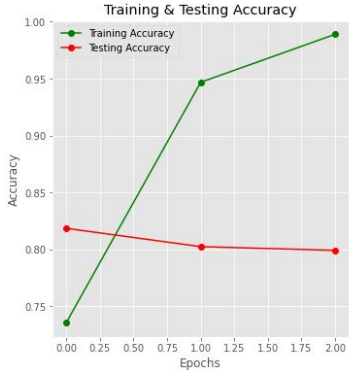

Figure 18: Train and test accuracy of LSTM-GRU with Word2Vec on news dataset

5.2.3 Sequential model with BERT

Using the sequential model and BERT, we managed to get a 92.37% accuracy on the news headline dataset, which is similar to the SVM and Keras models that were ran on this dataset. Ofcourse, the algorithm took much more time to run compared to the previous methods. The plot of the accuracy

is given in figure 19 .

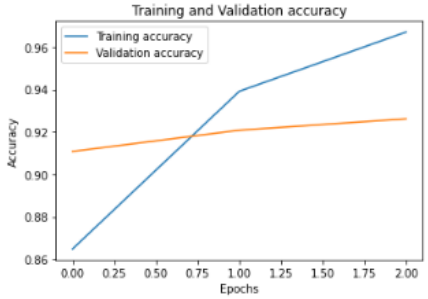

Figure 19: Training and validation accuracy of Sequential+BERT model on news dataset

We can see that using BERT did not change the accuracy by a large margin, and in fact it reduced it.

5.2.4 BiLSTM and GloVe

For the Reddit dataset, the Bidirectional LSTM achieved 74.5% accuracy on the validation set with 2 epochs. The training set and validation set accuracy plot for the LSTM model is shown in figure 20.

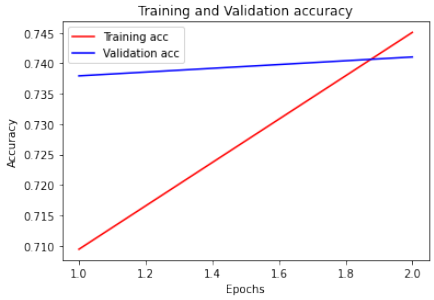

Figure 20: Training and validation accuracy of BiLSTM model on Reddit dataset

Using GloVe alongside the Bidirectional LSTM resulted in a noticeable boost for the accuracy of the model. While previous implemented LSTM achieved 74% accuracy, utilizing the GloVe embedding resulted in a 10% increase in terms of accuracy and performance of the model. In fact, the model achieved an accuracy of 84% on the test set.

5.2.5 Comparison

The next table compares the performance of different deep learning models that are used for different datasets.

Paper	Dataset	Model	Best Accuracy
(Salim et al., 2020)	Twitter	LSTM + Word embeddings	88%
(Ashok et al., 2020)	Twitter	BiLSTM + CNN + BERT	94%
(Shangipour ataei et al., 2020)	Reddit	BiLSTM + CNN + BERT	0.73 (F1)
Our Project	Reddit	BiLSTM	74%
		Sequential	67.21%
	News Headline	Sequential	93%
		LSTM-GRU + Word2Vec	80%
		Sequential + BERT	92%
		BiLSTM+GloVe	84%

6 Future Works

While we have proposed our formulation for mapping into the RL setting, we are still working on the technical aspects of the connection between the neural network and the RL agent. For future works, we would like to follow the proposed idea to implement RL-based sarcasm detection, which will hopefully enable us to apply it to similar tasks of text classification. We will also try to fine-tune our models on different datasets (such as Amazon and Twitter) and compare the results with the existing papers on those datasets.

7 Conclusion

As shown in the previous sections, several machine learning and deep learning models were implemented to address the problem of detecting sarcasm in a given sentence. We started to solve the problem by utilizing a vast range of algorithms, from machine learning to state-of-the-art deep learning approaches. Although sarcasm is a subjective matter, and one can call a sentence sarcastic while another person doesn't, we have obtained noticeable accuracy in both datasets. We tried to expand the classification task by utilizing knowledge graphs by doing some entity extraction and entity relation to do the classification task with more interpretability and potentially better results. We also tried to take advantage of Reinforcement Learning algorithms by mapping a formulation from the supervised setting into RL setting that is presented by utilizing Markov Decision Processes and Actor-Critic learning algorithm.

References

D. M. Ashok, A. Nidhi Ghanshyam, S. S. Salim, D. Burhanuddin Mazahir, and B. S. Thakare. 2020. Sarcasm Detection using Genetic Optimization on LSTM with CNN. In *2020 International Conference for Emerging Technology (INCET)*, pages 1–4.

J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.

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

Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, and Sune Lehmann. 2017. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1615–1625. ArXiv: 1708.00524.

Erik Forslid and Niklas Wikén. 2015. *Automatic irony- and sarcasm detection in Social media*.

Aditya Joshi, Pushpak Bhattacharyya, and Mark J Carman. 2017. Automatic sarcasm detection: A survey. *ACM Computing Surveys (CSUR)*, 50(5):1–22.

Akshay Khatri and Pranav P. 2020. Sarcasm detection in tweets with BERT and GloVe embeddings. In *Proceedings of the Second Workshop on Figurative Language Processing*, pages 56–60, Online. Association for Computational Linguistics.

Diana Maynard and Mark Greenwood. 2014. Who cares about sarcastic tweets? investigating the impact of sarcasm on sentiment analysis. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, pages 4238–4243, Reykjavik, Iceland. European Languages Resources Association (ELRA).

Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. 2021. Deep learning–based text classification: A comprehensive review. *ACM Computing Surveys (CSUR)*, 54(3):1–40.

Natalie Parde and Rodney Nielsen. 2018. Detecting Sarcasm is Extremely Easy ;-). In *Proceedings of the Workshop on Computational Semantics beyond Events and Roles*, pages 21–26, New Orleans, Louisiana. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Nela Petrželková, Blaž Škrlj, and Nada Lavrač. 2020. Knowledge graph aware text classification.

Ellen Riloff, Ashequl Qadir, Prafulla Surve, Lalindra De Silva, Nathan Gilbert, and Ruihong Huang. 2013. 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.

Sayed Saniya Salim, Agrawal Nidhi Ghanshyam, Darkunde Mayur Ashok, Dungarpur Burhanuddin Mazahir, and Bhushan S. Thakare. 2020. Deep LSTM-RNN with Word Embedding for Sarcasm Detection on Twitter. In *2020 International Conference for Emerging Technology (INCET)*, pages 1–4, Belgaum, India. IEEE.

Suyash Sangwan, Md Shad Akhtar, Pranati Behera, and Asif Ekbal. 2020. I didn't mean what I wrote! Exploring Multimodality for Sarcasm Detection. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, Glasgow, United Kingdom. IEEE.

Samer Muthana Sarsam, Hosam Al-Samarraie, Ahmed Ibrahim Alzahrani, and Bianca Wright. 2020. Sarcasm detection using machine learning algorithms in Twitter: A systematic review. *International Journal of Market Research*, 62(5):578–598.

Taha Shangipour ataei, Soroush Javdan, and Behrouz Minaei-Bidgoli. 2020. Applying transformers and aspect-based sentiment analysis approaches on sarcasm detection. In *Proceedings of the Second Workshop on Figurative Language Processing*, pages 67–71, Online. Association for Computational Linguistics.

Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Sen Wang, and Chengqi Zhang. 2018. Reinforced self-attention network: a hybrid of hard and soft attention for sequence modeling. *arXiv preprint arXiv:1801.10296*.

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Zelin Wang, Zhijian Wu, Ruimin Wang, and Yafeng Ren. 2015. Twitter sarcasm detection exploiting a context-based model. In *Web Information Systems Engineering – WISE 2015*, pages 77–91, Cham. Springer International Publishing.

Marco A Wiering, Hado van Hasselt, Auke-Dirk Pietersma, and Lambert Schomaker. 2011. Reinforcement learning algorithms for solving classification problems. In *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 91–96. IEEE.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. 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, San Diego, California. Association for Computational Linguistics.