Sarcasm Detection using a combination of LSTM, attention and manual features.

Mattia D'Agostini
Group project team members: Evangelia Bourazani,Ivan Samodelkin, Juan Pedro Danza Rovira
Advanced Natural Language Processing / Cognitive Systems / Department of Linguistics
University of Potsdam, Germany
March 2025
mattia.dagostini@uni-potsdam.de

Abstract

For the final project of the class ANLP W.S 2024, IsThisLoss [1], our team, decided to work on SemEval 2022's task 6, more specifically on subtask A, a sarcasm detection task. In this paper we aim to describe our methodology, experiments and results. Aiming to build a model that could perform well on the task while being low in computational cost, we implemented an LSTM-based model with an attention layer and combined with a series of manually extracted linguistic features, and an almost identical architecture based on a Bi-LSTM instead. We trained the models only on the task training dataset and tuned their hyperparameters with a Bayesian optimization algorithm. After tuning, we implemented variations of these models and evaluated them on the task test dataset. The two best performing models, the Bi-LSTM with attention and manually extracted linguistic features and the LSTM with attention and features, achieved a F_1 score of 0.264 and 0.262 respectively. This results showed the need of external dataset for training when it comes to sarcasm detection and the utility of manually extracted linguistics features, as models featuring them performed better when compared to models without them.

1 Introduction

In Wilson (2006), Deirdre Wilson defines sarcasm as a form of verbal irony that occurs when there is a discrepancy between the literal and intended meanings of an utterance. Sarcasm detection it is still a challenging but necessary task in the field of Natural Language Processing. Detecting sarcasm is crucial when dealing with sentiment analysis. Sarcastic sentences, by definition, have a different meaning than what is literally written and due to their usually irony-based nature, they are often difficult to detect if their context is not provided.

[1] https://gitup.uni-potsdam.de/samodelkin/ANLP-Sarcasm-IsThisLOSS

A sentence such as

> I really love Mondays.

may appear genuine and truthful if considered without any context. To most of us, though, the sentence itself appears to be sarcastic, as we tend to associate Monday with a stressful day that breaks the weekend relax. Nowadays, models are often trained with corpus from the internet, especially from social networks such as X (formerly known as Twitter). As described in Abu Farha et al. (2022), the presence of sarcasm on the internet can be disruptive for computational systems that need to perform NLP tasks such as opinion mining or sentiment analysis. Corpus from these sources tends to be lacking in context, as it frequently consists of small text chunks, that may require knowledge outside of their boundaries to be completely understood. In the experiment described throughout this paper, we tackled with the first sub task of task 6 of the 2022 SemEval competition (Abu Farha et al., 2022). The task required to build a model to detect sarcasm in tweets and human-generated text. We decided to implement 2 models, an LSTM with attention layer and manually extracted linguistic features, and a Bi-LSTM with attention layer and manually extracted linguistic features. Subsequently, we implemented 3 additional models based on variations on the first 2. We decided to work with RNNs instead of transformers or pre-trained language models, which appeared to be the main trend in model architecture for participants in the competition, as we aimed to build a model that could perform well without dealing with high computational cost. Despite being the best-performing models, transformers and pre-trained language models (which are usually based on LLMs) require a lot of computational resources to be trained compared to LSTMs, which, as shown in Clavié et al. (2021), leads not only to an higher ecological impact but also to lower accessibility.

2 Related Work

The idea of using an LSTM model with an attention layer for detecting sarcasm comes from Olaniyan et al. (2023), where the LSTM-based sarcasm detector was found to be the second best performing model in terms of both accuracy and F_1 score, with its performance exceeded only by a fine-tuned BERT model. The combination of a recurrent neural network and manually-extracted linguistic features was inspired from Pandey et al. (2021). Features implemented in the model mainly consists in a selection from Šandor and Bagić Babac (2024). Additional features were implemented by evaluating the presence of emojis, negations and intersections in the data (González-Ibáñez et al. (2011),Joshi et al. (2017),Zhang et al. (2016)).

3 Task Formalization

The task and the associated rules were defined by Abu Farha et al. (2022).The task goal consists in detecting if a given sentence is sarcastic. More specifically, the task requires to build a model that predicts probability associated with the input being respectively in two classes, the sarcastic one and the non-sarcastic one. To achieve this, a labeled training dataset is given, alongside with a labeled test dataset for model evaluation. It is allowed to train the model on any dataset, even external ones, as long as it does not feature example from the test dataset in it.

4 Data

Training and testing data comes from task 6 of 2022 SemEval competition (Abu Farha et al., 2022). Our work has focused on sub-task A of this competition, being "Sarcasm Detection", and we decided to work with the english dataset. Training data featured in this dataset consists of tweets, annotated by their speakers and of labels identifying tweets as either sarcastic or non-sarcastic. Test data consists of sarcastic sentences made up by authors recruited by the competition organizers and of non-sarcastic sentences imported from other datasets. It is worth to note that a great portion of tweets features either hashtag or URLs. As our model works with embedded sequences of words using a pretrained Glove model (Pennington et al., 2014), both hashtag, @s mentioning users and URLs, due to their uniqueness, are not possible to embed. This has led to tweets portions being unusable, and in case of some tweets, consisting merely of URLs or hashtag, the whole tweet was removed during pre-processing from the training set as, after pre-processing, it would result in an empty tensor. In many cases, even though the whole tweet was not deleted, the remaining part after pre-processing was found to have lost most of the original text meaning, such as in the following example

> @BorisJohnson @MattHancock Oh really?

As it is evident, the sarcastic nature of this tweet can be found in the addressed people, being British politicians. Without the knowledge of the users mentioned, it would be really difficult even for an human to determine wether the sentence

> Oh really?

is sarcastic.

5 Methodology and Experiments

5.1 Data extraction and pre-processing

First, tweets and labels are extracted from both the train and test CSV files. Tweets are stored in the CSV files as strings, encoded in UTF-8 standard. During pre-processing, each tweet goes through the following stages

1. Tweet is converted to lowercase
2. Punctuation is removed from the tweet
3. Stop-words are removed from the tweet
4. Tweet words are lemmatized
5. Hashtag are removed

Pre-processing is done in order to standardize the input. On one side, this approach makes it easier to embed tweets word by word, and, by reducing the vocabulary with lemmatization, the model can recognize specific verbs or phrase structures more easily, as differences in verb conjugations or unuseful words, such as stop words, are removed during this phase of the experiment. The set of stop words used for pre-processing comes from the NLTK module (Bird and Loper, 2004), and it is the standard stop-words set for english language for this specific library. The lemmatizer used is the WordNetLemmatizer, which is also from the NLTK library.

5.2 Embeddings

Pre-processed tweets are tokenized by simply splitting each tweet when a space occurs, and are embedded into tensors of size 100 using a pre-trained Glove model (Pennington et al., 2014) specifically trained on tweets. The pre-trained model can be downloaded for free from the Glove project website[2]. The embedding size was taken from Pandey et al. (2021). The choice was motivated by the fact that 100 is an intermediate embedding dimension, and, considering the fact that the training dataset consisted of roughly 3500 tweets, choosing a smaller embedding size would have resulted in repetitive and inefficient embeddings, meanwhile an higher one would have created a too sparse representation of the data. As previously mentioned, some tweets contained in the data featured URLs or mentions towards other users (represented by the @ symbol followed by the user name). This particular part of data could not be embedded, as neither an URL or an username can be recognized by an embedder that has not been trained on it, and often embedder developers avoid training them on this kind of data as it is unique and does not fall into the usual definition of word that embedders are built to deal with. As previously mentioned this has led to some tweets being left out from both training and testing or being represented with just few words that were not able to convey the original tweet meaning

5.3 Attention Layer

The choice of implementing an attention layer in the LSTM and Bi-LSTM models was made in reference to the classifiers implementations seen in Šandor and Bagić Babac (2024) and (Olaniyan et al., 2023). We choose to implement a soft-attention layer in Pytorch and we included in most of models used during the experiment. Among the available variations, we decided to use a soft attention mechanism as it was featured in sarcasm-related literature as (Pandey et al., 2021) and (Jain et al., 2020). The LSTM layer in our model processes the embedded input producing an hidden state h_t for every sequence step t. Each hidden state is produced after computing a specific word of the tweet and it represents relevant information captured by the layer. The result of this process is a collection of hidden states $\{h_1, \ldots, h_n\}$. For every hidden state h_i, the attention layer computes a score using the following equation

$$e_i = v^\top \tanh(W h_i) \tag{1}$$

where $W \in \mathbb{R}^{(d \times d)}$ and $v \in \mathbb{R}^d$ are learnable parameters. A softmax function is then responsible for normalizing the obtained scores, resulting in attention weights

$$a_i = \frac{\exp(e_i)}{\sum_{j=1}^{n} exp(e_j)} \tag{2}$$

Finally the context vector c is computed as the weighted sum of the hidden states

$$c = \sum_{i=1}^{n} a_i h_i \tag{3}$$

The context vector is used as an additional input by the decoder, which allows the classifier to focus on the most relevant part of the extracted information.

5.4 Feature engineering

Alongside of the recurrent neural network, we defined a series of manual features to extract from each input. The idea of combining machine learning with manually extracted linguistic features was implemented by Šandor and Bagić Babac (2024) combined with a bi-LSTM. A similar approach to the one presented in this paper can be found in Pandey et al. (2021),where manually extracted features are used in combination with an LSTM with an attention layer. All features implemented are Boolean features, representing with a binary output the presence of a certain linguistic characteristic in a given tweet. Detectors for the following linguistic features were implemented in our model

1. Emoji
2. Repeated letters
3. Presence of question/exclamation mark
4. Repeated punctuation
5. Three dots
6. Quotation
7. Negations
8. Interjections

[2] nlp.stanford.edu/projects/glove/

Features 2-7 where used in Šandor and Bagić Babac (2024), Emoji detection in González-Ibáñez et al. (2011), presence of negations as an indicator of sarcasm is from Joshi et al. (2017) and interjections detection was featured in Zhang et al. (2016). A selection of the most relevant features is analyzed in the following paragraphs.

5.4.1 Emojis

Emoji detection was inspired by González-Ibáñez et al. (2011), who stated that emojis and emoticons can be considered pragmatic features to detect positive or negative emotions, hence being indicative of sarcasm. Evidence supporting this statement was found on the dataset, where 21% of sarcastic tweets featured emojis.

5.4.2 Presence of question/ exclamation mark

The presence of question or exclamation marks as an indicator of sarcasm was featured in Šandor and Bagić Babac (2024), in which authors pointed out that sarcastic comments tend to have more punctuation, often used to stress the emotion they're expressing through the sentence (e.g "I adore being bored!!!"). In the dataset, 23.30% of sarcastic tweets featured question or exclamation marks.

5.4.3 Repeated punctuation

For the same reasons as question and exclamation marks, Šandor and Bagić Babac (2024) assumed that the repeated use of punctuation could be a valid indicator of sarcasm. In the data set 6.34% of sarcastic tweets featured repeated punctuation.

5.4.4 Presence of three dots

Being a particular instance of punctuation, the logic applied to this feature was similar to the repeated punctuation one. The difference that made us believe that it would deserve its own feature was the fact that often in tweets the three (or more) dots are used to express a suspended thought, which is usually associated with sarcasm. in the data set, 5.07% of sarcastic tweets featured three dots.

5.4.5 Presence of negations

The presence of negations as an indicator of non-sarcasm was featured in Joshi et al. (2017). For detecting negations, we used the set of negation words from Padmaja et al. (2014). In the dataset, 19.15% of non-sarcastic tweets featured negations.

5.4.6 Presence of interjections

The presence of interjections as an indicator of sarcasm was featured in Zhang et al. (2016). For interjection we intend a word that is used to express a feeling without having strong grammatical connections to other sentence elements. Interjections are often found isolated and tend not to combine with other words. The set of interjections used for detection can be found on the Enchanted Learning website [3]. In the dataset, 66.32% of sarcastic tweets featured interjections.

Linguistic feature	Tweets percentage
Emoji	20.99
Repeated letters	11.53
Question/Exclamation	23.30
Repeated punctuation	6.34
Three dots	5.07
Quotations	19.26
Interjections	66.32

Table 1: Features distribution in dataset

After being computed, the feature vector, consisting of binary entries for representing the presence of each feature in a given tweet, is converted to a tensor and used as part of the model input.

5.4.7 Discarded features

In our project presentation we originally proposed three additional features

1. Hashtags detection
2. Incongruity detection
3. Hyperbole detection

Even though all of them had supporting literature, we found out that the distribution of those feature in our particular case, considering the available training dataset was not ideal, as all three features were found more often in non-sarcastic tweets rather than in sarcastic ones.

Linguistic feature	% of N.S.T	% of S.T
Hashtags	18	14
Incongruity	4.0	3.4
Hyperbole	21.7	20.9

Table 2: Discarded features distribution in dataset (N.S.T stands for non sarcastic tweets and S.T for sarcastic tweets)

[3] https://shorturl.at/2uEK8

5.5 Model Architecture

The main model architecture used in our experiments consists in a combination of an LSTM with an additional attention layer and a series of manually-extracted linguistic features, combined with the LSTM output and processed by a dense layer returning the probability associated with the input being sarcastic or not in form of a binary classification. The architecture was mostly based on Pandey et al. (2021), variations were mainly made on the coding side as the paper in question did not disclose details about the programming part of the experiment.

5.5.1 LSTM layer

The implementation for the LSTM layer was done by defining a model class in Pytorch. The specifics for the implementation were taken from Pandey et al. (2021). More specifically, the LSTM layer of our classifier consists of 2 concatenated LSTM layers, the second one having half the hidden size of the first one. During the model tuning we ran multiple instances of training by varying this parameter, but the ratio between the first and the second LSTM layer was never changed. The custom Pytorch implementation was necessary not only to deal with the different hidden sizes of LSTM layers, but also to make sure that the model could work with packed inputs, as most of the training occurred with batch sizes $\neq 1$ to avoid overfitting. In the final implementation, the LSTM layer receives the embedded input and returns the collection of hidden states $\{h_1, \ldots, h_n\}$ produced at any sequence step $n \geq 1$.

5.5.2 Attention Layer

The LSTM output is subsequently processed by an attention layer, which architecture has been described in the dedicated paragraph above. The attention layer takes the collection of hidden states $\{h_1, \ldots, h_n\}$ produced at any sequence step $n \geq 1$ and returns the context vector associated with it, which, predictably, has the same dimension as the hidden state of the last LSTM layer. The attention layer returns the weights associated with the context computation as well but we ended up not using them for deeper model analysis.

5.5.3 Dense Layer

After being computed, the context tensor is concatenated over the last dimension to the extracted features tensor, producing a tensor of size $z = \frac{h}{2} + 8$, where h is the hidden size of the first LSTM layer. The resulting tensor is passed through a dense layer of size $(z, 2)$, which produces a tensor representing the probability distribution for each classification class, where the first class is associated with the input being non-sarcastic and the second one with it being sarcastic. During training the model output consists of the raw probabilities, meanwhile in testing, only the index of the highest probability is returned, sharing only the class predicted by the model and not details regarding the prediction.

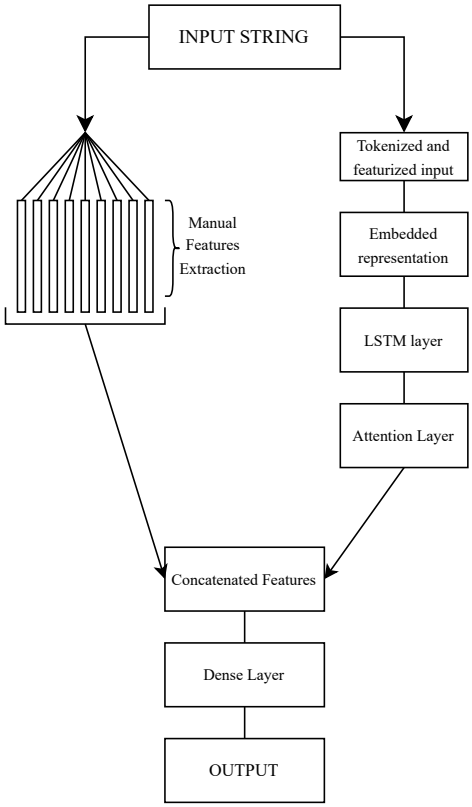

Figure 1: Main model architecture

5.5.4 Architecture variations

During our experiment we implemented 4 variations of the main model, in order to explore if different layers combinations could lead to notable results. More specifically, we implemented a Bi-LSTM with attention layer and features classifier, where the two LSTM layers were replaced by a number Bi-LSTM layer (the layer number parameter was explored during the tuning phase of this model), an LSTM model without attention and features, an LSTM model with attention and without features and a simple LSTM.

5.6 Metrics

To evaluate the model we used the following metrics.

5.6.1 Accuracy

Accuracy is defined as the number of correct predictions made by the model over the total number of predictions. It can be a good metric to compute performance, but has its limits as if there's an imbalance in distribution between classes in the dataset, guessing randomly can lead to a good accuracy even though the model is not performing well. Accuracy uses the following formula

$$Accuracy = \frac{T_p + T_n}{T_p + T_n + F_p + F_n} \tag{4}$$

where T_p and T_n represent respectively true positive and true negative predictions, while F_p and F_n represent false positive and false negative predictions (where positive is intended the sarcastic class).

5.6.2 F_1 score

F_1 score is defined as a combination of precision and recall through the following equation

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{5}$$

Where Precision is defined as

$$Precision = \frac{Tp}{T_p + T_n} \tag{6}$$

And recall as

$$Recall = \frac{T_p}{T_p + F_n} \tag{7}$$

The F_1 provides more insight into the model performance, as it requires good scores in both sarcastic and non-sarcastic classification to be high.

5.7 Training and tuning

We trained our models using an Adam optimizer (Kingma and Ba, 2017), with varying learning rate and a weight decay value of 10^{-5}. We used cross entropy as loss function (Jurafsky and Martin, 2025), which has the following general equation

$$L = -\sum_{k=1}^{K} y_k \log(p_k) \tag{8}$$

where y_k is the correct label and p_k is associated to each class k of the classifier (in this case they were 2).

5.7.1 K-fold cross validation

K-fold cross validation (Kohavi et al., 1995) is a technique used to judge the model performance in a more detailed way rather than just evaluating its performance on the test set. Using this procedure, the training set it's split in k parts, and k iterations of training are run (called folds). During the i-th fold, the i-th portion of the training dataset is not used for training, but rather used as a test set for evaluating the model performance, called evaluation set. Given that at every fold a new instance of the model is initiated, the model configuration is evaluated k times, every time with a slightly different training set, and a different validation set. In our experiment we decided to use a 5-fold cross validation technique, which we estimated as a good balance between the number of model instances evaluated and the computational cost, as using k-fold cross validation requires a training of time of $k * t$, where t is the time required for a single training instance.

5.7.2 Tuning

Tuning was done using a Weights & Biases sweep instance (Biewald, 2020), allowing us to loop through different hyperparameters configurations and compute the best ones. The best models were chosen by averaging the validations loss (cross entropy loss computed on the validation set) across the 5 training folds. The sweep was conducted with a Bayesian optimization search algorithm (Snoek et al., 2012) included in Weights & Biases, that allowed us to determine the optimal model configuration without computing all possible combinations. Hyperparameters involved in the sweeping are reported in the table below.

Parameter	Value
Learning rate	$10^{-3}, 10^{-4}, 10^{-5}$
Epochs	$100, 500, 1000$
Hidden size	$128, 256, 512$
Number of hidden layers	$1, 2$
Batch size	$16, 32, 64$

Table 3: Tuning parameters

It is important to note that, as previously mentioned, the number of layers in the tuning hyperparameters refers only to the Bi-LSTM-based architectures, as the LSTM-based architectures had their own double layer architecture that was not changed during tuning. Alongside to this information, the parame-

ter "Hidden size" size when tuning an LSTM-based model, refers to the first LSTM layer (the second one, as mentioned, has half the size of the first one), meanwhile when tuning a Bi-LSTM-based model it refers to any bi-LSTM layer featured in that specific architecture.

6 Results

6.1 Tuning results

After tuning both the Bi-LSTM and LSTM-based models, the optimal configuration hyperparameters found by the Bayesian algorithm were the following

Parameter	LSTM	Bi-LSTM
Learning rate	10^{-4}	10^{-4}
Epochs	100	100
Hidden size	128	128
Number of hidden layers	1	∅
Batch size	32	32

Table 4: Optimal configuration for Bi-LSTM/LSTM with attention and features

An higher learning rate was found to be leading to overfitting, while any learning rate smaller than 10^{-4} led to the model labeling all of the examples as non-sarcastic. We trained the following models using the optimal hyperparameters

1. LSTM
2. LSTM with attention
3. LSTM with manually extracted features
4. LSTM with attention and manually extracted features
5. Bi-LSTM with attention and manually extracted features

6.2 Models performance

Models performance was evaluated on the test set ranked according to the F_1 score obtained.

Model Architecture	F_1 score
Bi-LSTM with attention and features	0.264
LSTM with attention and features	0.262
LSTM with attention	0.261
LSTM	0.247
LSTM with features	0.230

Table 5: F_1 test score by different model architectures

The best performing model was found to be the Bi-LSTM with attention and manually extracted features. This result, in spite of the lower score, is similar to what (Pandey et al., 2021) and (Šandor and Bagić Babac, 2024) had shown, with manual features, added in combination with an attention layer, led to an increase in model performance. The model performed well in detecting non-sarcastic inputs, but lacked in detecting sarcastic ones, lowering the final F_1 score.

Metric	Value
Accuracy	0.74
Precision	0.22
Recall	0.32
F_1-Score	0.26

Table 6: Evaluation Metrics for Sarcasm Detection for the Bi-LSTM model with attention

The results align with the rankings presented in Abu Farha et al. (2022), where models different than transformers trained exclusively on the training data performed with similar results. A deeper look into results (the full results table is available in the appendix) shows reasons for the low score. The low F_1 is mainly caused by the model predicting most of the inputs as non-sarcastic. As it can be seen in the confusion matrix (available in the appendix), out of 1397 inputs, the model labeled 1112 of those as non-sarcastic, resulting in just 64 true positives and 136 false negatives. The model, given the fast pace at which Bi-LSTM-based models learn, and the small dataset available for training, even with few training epochs reaches scores close to over-fitting when evaluated over the training data. Both accuracy and F_1 score are significantly higher when the model is evaluated on the training data. The main causes for scores being close to over-fitting are, in our opinion, the lack of training on external datasets, implemented by most of the competition participants. However, we're sure that there would still be room for improvement if the model was trained with a wider range of hyperparameters or external datasets, which we ended up not doing due to the required computational cost and our limited resources. More about the dataset can be found in the qualitative analysis section.

6.3 Qualitative analysis

One of the reasons behind the low F_1 score could be lying in the test data. While the training data

was made of tweets collected from different accounts by Abu Farha et al. (2022), the test dataset was collected by the organizers of the competition by asking people to write examples of sarcastic sentences. The different origin of the two dataset may have led to different sarcasm patterns, as language is heavily influenced by context.
The best performing model, being the Bi-LSTM with attention and manually extracted features presented some notable patterns when classifying data. The model often categorized non-sarcastic sentences that featured strong words or strongly subjective words as non-sarcastic. False positives were reported also when the model was asked to classify sentences about politics (most of them being about Brexit) and the pandemic. The model incurred in false negative classifications mostly in tweets about Christmas. This was most probably caused by the presence of multiple non-sarcastic Christmas-related tweets in the training dataset as opposite to the test dataset where some instances of Christmas-related sarcastic tweets were featured. A table with notable examples is available in the appendix.

7 Conclusion

During the experiment, we built two main models, an LSTM with an attention layer and manually extracted linguistic features, and a Bi-LSTM with an attention layer and manually extracted linguistic features. We then tuned these models over the Abu Farha et al. (2022) dataset to retrieve optimal hyperparameters for training. After finding the optimal hyperparameter configuration, we trained three additional variations of the LSTM model, reaching a total of 5 implemented sarcasm detectors. The best performing model was the Bi-LSTM with an attention layer and manually extracted features, achieving a F_1 score of 0.26 on the test dataset. By analyzing our results, we suggest that this score could be improved by training with additional datasets . Moreover, it would be insightful to train the model with more hyperparameters. Unfortunately, due to computational cost and time contingencies, we were not able to explore this aspect of the experimentation as much in depth. Similarly, we expect the score to improve by implementing more linguistics features, as they have shown to produce an increase in terms of performance when applied.

8 Limitations and Ethical considerations

8.1 Limitations

The main limitation we encountered during the experiment has been limited computational resources. Tuning a Bi-LSTM, when the hidden size grows, may take up to 4 hours per model, due to our limited resources. On this basis, we decided to train the models on a Google Cloud Virtual Machine, which allowed us to have access to a GPU instead of relying on our laptop CPUs to perform training. Due to some technical issues, and to our lack of knowledge in Weights & Biases using, we ended up spending most of our training credits in tuning two models only, which made our exploring of all possible model configurations limited.

8.2 Ethical considerations

8.2.1 Dataset

Both the training and test dataset were provided by Abu Farha et al. (2022), that collected both sarcastic and non-sarcastic tweets from authors recruited using the Prolific Academic[4] platform. For the test set, native english speakers were recruited using the Appen [5] crowdsourcing platform and they were asked to write a short sarcastic text on the spot. In the paper is not specified how non-sarcastic test tweets were obtained.

8.2.2 Environmental considerations and accessibility

We decided to implement an LSTM-based model rather than using a pre-trained model language (which are usually LLMs-based) to avoid using too much computational power for a simple project. As showed in Clavié et al. (2021), even though transformers-based model perform really well in classification tasks, they require a great amount of computational power, which leads to a higher ecological impact and lower accessibility. We believe that technology should be sustainable ad accessible, and we aim to develop better tools in the future to reach this results.

8.2.3 Decisions about truth value

Classifying a text input into a sarcastic or non-sarcastic category is a decision about its truth value. A tool that can infer wether a piece of text is describing the truth or presenting a false sentence with a sarcastic intent is a tool that shall be used

[4]https://prolific.co
[5]https://appen.com

with responsibility, as it can ran easily into biases, as shown by our qualitative analysis.

References

Ibrahim Abu Farha, Silviu Vlad Oprea, Steven Wilson, and Walid Magdy. 2022. SemEval-2022 Task 6: iSarcasmEval, intended sarcasm detection in English and Arabic. In *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*, pages 802–814, Seattle, United States. Association for Computational Linguistics.

Lukas Biewald. 2020. Experiment tracking with Weights and Biases. Software available from wandb.com.

Steven Bird and Edward Loper. 2004. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 214–217, Barcelona, Spain. Association for Computational Linguistics.

Benjamin Clavié, Akshita Gheewala, Paul Briton, Marc Alphonsus, Rym Laabiyad, and Francesco Piccoli. 2021. Legalmfit: Efficient short legal text classification with lstm language model pre-training. *Preprint*, arXiv:2109.00993.

Roberto González-Ibáñez, Smaranda Muresan, and Nina Wacholder. 2011. Identifying sarcasm in Twitter: A closer look. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 581–586, Portland, Oregon, USA. Association for Computational Linguistics.

Deepak Jain, Akshi Kumar, and Geetanjali Garg. 2020. Sarcasm detection in mash-up language using soft-attention based bi-directional LSTM and feature-rich CNN. *Applied Soft Computing*, 91:106198.

Aditya Joshi, Pushpak Bhattacharyya, and Mark J. Carman. 2017. Automatic sarcasm detection: A survey. *ACM Comput. Surv.*, 50(5).

Daniel Jurafsky and James H. Martin. 2025. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*, 3rd edition. Online manuscript released January 12, 2025. https://web.stanford.edu/~jurafsky/slp3.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization. *Preprint*, arXiv:1412.6980.

Ron Kohavi and 1 others. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada.

Deborah Olaniyan, Roseline Oluwaseun Ogundokun, Olorunfemi Paul Bernard, Julius Olaniyan, Rytis Maskeliūnas, and Hakeem Babalola Akande. 2023. Utilizing an Attention-Based LSTM model for detecting sarcasm and irony in social media. *Computers*, 12(11):231.

S Padmaja, S Sameen Fatima, Sasidhar Bandu, and B. Sowmya. 2014. Comparison of the scope of negation in online news articles. In *International Conference on Computing and Communication Technologies*, pages 1–6.

Rajnish Pandey, Abhinav Kumar, Jyoti Prakash Singh, and Sudhakar Tripathi. 2021. Hybrid attention-based long short-term memory network for sarcasm identification. *Applied Soft Computing*, 106:107348.

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.

Daniel Šandor and Marina Bagić Babac. 2024. Sarcasm detection in online comments using machine learning. *Information Discovery and Delivery*, 52(2):213–226.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of machine learning algorithms. *Preprint*, arXiv:1206.2944.

Deirdre Wilson. 2006. The pragmatics of verbal irony: Echo or pretence? *Lingua*, 116(10):1722–1743. Language in Mind: A Tribute to Neil Smith on the Occasion of his Retirement.

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, Osaka, Japan. The COLING 2016 Organizing Committee.

A Appendix

Metric	Value
Validation Accuracy	0.701881
Validation Precision	0.340580
Validation Recall	0.290123
Validation F_1 score	0.313333
Training Accuracy	0.958379
Training Precision	0.963780
Training Recall	0.869318
Training F_1 score	0.914115

Table 7: Additional Evaluation Metrics for Sarcasm Detection for the Bi-LSTM model with attention

Example	Category	Reason of Interest
Having a really busy day, might only manage the one box set today.	False Negative	Lack of context
My heart. Loving the music, the dancing. Loving life.	False Positive	Strong subjective language
Very very happy!!!	False Positive	Strong subjective language
Well done everyone you all did fantastic. clapemoji x3	False Positive	Strong subjective language
I really wish people would stop being selfish and wear a mask in shops. Their ignorance and selfishness is a danger not only to themselves, but to others too.	False Positive	Covid
Please wear your mask, and please stay at home if you are coughing and spluttering, you could give someone COVID, such as myself.	False Positive	Covid
21:56 Temp. 4°C, Hum. 92, Dewp. 2.4°C, Bar. 989.6 hpa, Rain Today 2.9000 mm, Wind 54° 2.9 kmh	False Positive	Weather
09:53 Temp. 5.5°C, Hum. 77, Dewp. 0.8°C, Bar. 996.6 hpa, Rain Today 2.2000 mm, Wind 5° 1.4 m/s	True Negative	Weather
Boris and his government team are pathetic.	False Positive	Brexit
The PM's 'Plan B' announcement is obviously a diversionary tactic	False Positive	Brexit
I think Boris Johnson deserves his parties after the great job he does for the country these past few years	True Positive	Brexit
How kind of the Conservative party to cancel their Christmas get together this year, I mean they didn't have enough parties last year, did they?	False Negative	Christmas
I'm soo excited to cook a three course Christmas dinner for my whole family this year!	False Negative	Christmas

Table 8: Examples of model classification errors

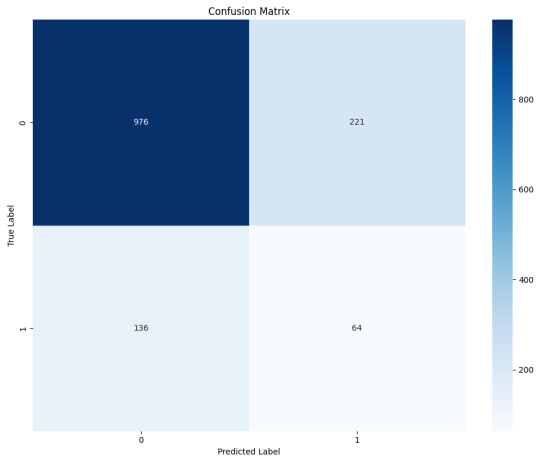

Figure 2: Confusion matrix for model evaluation on the test set

A.1 Personal considerations

During this project, I had the chance to work in a really enjoyable environment and with awesome people. We supported each other a lot during these two months, filling each other's gaps and bringing each one their background to the table. I was responsible for part of the coding, helping Ivan in building the models and building scripts for training, testing and utilities. I learned how to set up a VM and how to code in a more efficient and understandable way. Moreover, i took part in results discussion with the rest of the group. During this project I learned how coding needs to be on point when working in groups, as bugs or required implementations need to be addressed by everyone and anyone should be able to understand code written by other group members. During this semester I fell in love with the subject of linguistics, which I never had the occasion to study before in academical contexts. At first, my lacking in linguistics knowledge, due to my mathematical background, scared me and made me feel insecure. But after months of classes and time spent with my colleagues (which I'd now define as friends) I feel like I'm starting to understand more the importance of this subject and the true nature of Natural Language Processing. This project made me face my weaknesses, being mostly working in group and writing in Academical English about scientific topics and I'm happy for that, as now I'm less afraid of them. At the end of the semester, I can say that I improved a lot thanks to the amazing connections I made along the way and I can't wait to improve more.