IDA - Department of Computer and Information Science
Linköping University

Impact of different text vectorizations and embeddings in sarcasm identification

732A92 - Text Mining - Project Report

Marcos Freitas Mourão dos Santos
LiU ID: marfr825

Sweden
March 11, 2021

Abstract

Converting text into numerical representations is a key part of Natural Language Processing (NLP). Vectorization methods can vary in complexity, and some capture linguistic features that others do not, such as contextual differences of words. Sarcasm identification in text is a highly context-dependent task where positive words can have negative connotations and *vice-versa*. This report focused on the comparison of different word vectorization techniques, namely Term Frequency - Inverse Document Frequency (TF-IDF), pre-trained Continuous-Bag-of-Words (CBOW) and distilled Bidirectional Encoder Representations from Transformers (distilBERT) embeddings and their sarcasm classification accuracy using a Linear Support Vector Classifier (L-SVC) against a random choice baseline. The results show that sarcasm classification benefits from context-sensitive embeddings: The baseline, TF-IDF, CBOW reached, respectively, 50%, 79% and 81% accuracy on test data. Meanwhile, distilBERT embeddings achieved 86% accuracy. The analysis had limitations as embeddings were not fine-tuned for the task at hand.

Contents

1 Introduction

The Cambridge Dictionary of English defines sarcasm as "the use of remarks that clearly mean the opposite of what they say, made in order to hurt someone's feelings or to criticize something in a humorous way". From a human perspective, the detection of sarcasm in speech is natural to most. However, human interaction involves much more than the words themselves: volume, tone, context, etc. It can be argued that sarcasm in text form is more difficult to be detected even by humans as many of the former social cues are absent.

From the natural language processing (NLP) perspective, sarcasm is a challenging problem, specially for sentiment analysis (Patro et al., 2019). This is partially due to the nature of sarcasm itself: positive words with negative connotation and *vice versa*, and partially to the lack of good labelled datasets (González-Ibánez et al., 2011). Sarcasm identification tools are of interest for companies and individuals alike as a mean of correctly identifying one's target audience true opinions. It is desired, therefore, to deploy efficient models to correctly classify sarcasm.

The very first step into implementing a text classifier is converting raw text into a numerical representation that can be interpreted by a computer, i.e. a numerical vector. There are several ways of achieving this.

A simple, common approach is to compute Term Frequency–Inverse Document Frequency (TF-IDF) of a corpus. Each term is assigned a weight based on its frequencies in documents and in corpus. It is also known as a *bag-of-words* model, as word order, context and semantics do not matter (Manning et al., 2008).

A more sophisticated option is the Word2Vec model, in particular the Continuous-Bag-of-Words (CBOW) architecture, a neural network designed to predict a current word based on surrounding words, i.e. context (Mikolov et al., 2013). This model implements word embeddings so that semantic relations between words are preserved (Mikolov et al., 2013). Although the word *context* is used here, this model does not capture contextual embeddings, i.e. it only learns static vector representation for each word, independent of context (Ethayarajh, 2020). As an example, the word *apple* in the sentences "I ate an apple" and "I wish I had an Apple computer" would have the same vector representation, even though they mean completely different things.

More recently, transformer architectures have achieved impressive results due to its parallelization capabilities (Vaswani et al., 2017). Bidirectional Encoder Representations from Transformers (BERT) uses a bidirectional transformer to pre-train text representations that are context-sensitive, unlike Word2Vec (Devlin, 2019). In the previous example, the word *apple* would have two different vector representation: one representing the food and another representing the tech company.

Given this, it seems reasonable to think that context-sensitive models would outperform context-insensitive models in sarcasm detection. These models will be discussed in more detail in the following section.

The aim of this work is not to achieve state-of-the-art classification metrics, but rather investigate different text vectorizations and their impact on classification performance of sarcasm. This project was chosen as a compliment to the Text Mining course content due to its interesting nature and to have an introductory contact with more complex word embeddings/language models.

2 Theory

2.1 Term Frequency-Inverse Document Frequency

TF-IDF, as the name suggests, has two components: Term Frequency and Inverse Document Frequency and both are weights assigned to a given term. The former, as Manning et al. (2008) defines it, is "equal to the number of occurrences of term t in document d", or simply denoted by $\mathrm{tf}_{t,d}$. The latter, on the other hand, is weight designed to attenuate the *tf* score of overly common words that possibly have no meaningful importance given a set of documents, e.g. the word *game* in a collection of video game reviews has no impact in discriminating these reviews from one another: Is it talking about *Super Mario* or *Tetris*? IDF is defined by Equation 1.

$$idf_t = \log \frac{N}{df_t} \tag{1}$$

Where N is the total number of documents and df_t is the term's document frequency, i.e. the number of documents that term is in.

Finally, TF-IDF is defined in Equation 2 as the product of TF and IDF (Manning et al., 2008). It assigns to each term in a corpus a value that indicates how "important" this word is based on the frequency of said term in each document, TF, and how rare documents containing said word are, IDF (Manning et al., 2008). This method, however, has limitations as it only takes into account term frequencies; it does not capture semantic similarities between words. In addition to that, the TF-IDF representation is normally large and sparse, which can be troublesome for huge corpora (Kuhlmann, 2020).

$$\text{tf-idf}_t = \mathrm{tf}_{t,d} \times \mathrm{idf}_t \tag{2}$$

2.2 Word2Vec: Continuous-Bag-of-Words

The main goal of Word2Vec is to produce numerical representations of words such that similar words are closer to each other and dissimilar words are far apart. There are two models inside the Word2Vec family: Continuous-Bag-of-Words (CBOW) and skip-gram. This report will focus on the former [1]

The main objective of CBOW is to predict a target word w_t based on surrounding words in a given window, called context words $w_{t-m}, w_{t-m+1}, ..., w_{t-1}, w_{t+1}, ..., w_{t+m-1}, w_{t+m}$, where m is the windows size.

The context words are first encoded into one-hot vectors $x_{t-m}, x_{t-m+1}, ..., x_{t-1}, x_{t+1}, ..., x_{t+m-1}, x_{t+m}$. The dimension of each vector is $[d \times 1]$ where d is the number of words in the vocabulary.

[1]Note: this theory explanation was based on (Ghodsi, 2017a) and (Ghodsi, 2017b).

Next, all context words are averaged by a matrix os weights $\mathbf{W}$ such that a overall context vector v_c is extracted.

$$v_c = \mathbf{W}^T \mathbf{x} \tag{3}$$

The context vector v_c, sequentially, gets projected back into a d dimensional space via a different weight matrix $\mathbf{W'}$. This new vector is named u.

$$u = \mathbf{W'}^T v_c \tag{4}$$

Each entry of this vector, u_i can be written as the product of a word vector v_w times the context vector v_c:

$$u_i = v_w^T v_c \tag{5}$$

Finally, the vector u is fed to a softmax layer where the output vector y is obtained, which is a probability vector such that it respects Equation 6. Each entry y_i is the probability of a word given the context as seen on Equation 7.

$$\sum_i y_i = 1 \tag{6}$$

$$y_i = p(w|c) = \text{softmax}(u_i) = \frac{e^{v_c^T v_w}}{\sum_w e^{v_c^T v_w}} \tag{7}$$

The training objective is to find v_w, v_c such it maximizes the log-likelihood of

$$L(v_w, v_c) = \sum_w v_c^T v_w - \log \sum_w e^{v_c^T v_w} \tag{8}$$

The model can be summarized in Figure 1.

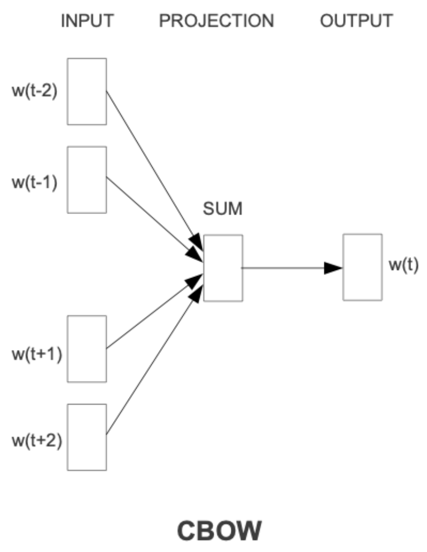


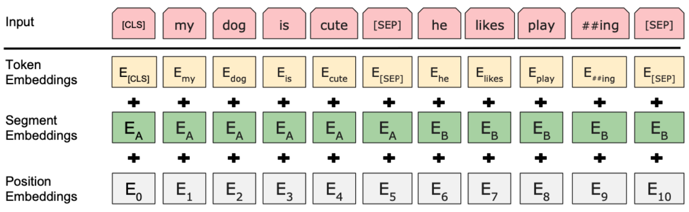
Figure 1: CBOW model (Mikolov et al., 2013).

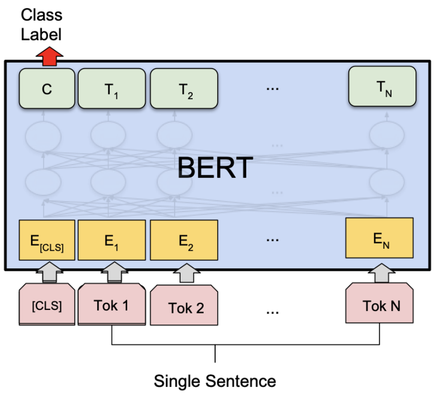
2.3 (Distil)BERT for single sequence classification

Transformer networks took the world by storm due to its paralellization capabilities and toppled established architectures such as LSTMs with new state of the art scores in several NLP tasks. While a traditional transformer has encoder and decoder blocks, Bidirectional Encoder Representation from Transformers (BERT), as the name suggests, only has encoder blocks (Halthor, 2020). The BERT architecture is pre-trained with huge corpora such the BookCorpus (800M words) and English Wikipedia (2,500M words) (Devlin et al., 2019).

BERT uses two techniques simultaneously for pre-training: Masked Language Model (MLM) and Next Sentence Prediction (NSP). The former consists into masking out a certain amount of words from sentences and training to predict those words, e.g. "The corona [MASK] outbreak" and the prediction of the word "virus". The latter is the task of judging if a sentence is a continuation of a previous sentence, e.g. The sentence "Voted in Nicaraguan elections" does *not* follow the sentence "The penguins of Antartica." The first task is responsible to train context *within* the sentence, while the second task trains context between sentences.

Every sequence in BERT has the special [CLS] token as the first token (see Figure 2a). Additionally, every BERT input embedding, E, is the sum of three different embeddings: token, segment, and position. The token embedding is retrieved from the pre-trained WordPiece embeddings, with a 30,000 token vocabulary. The segment embeddings encode which sequence it is dealing with (sequence A or B in a NSP task). Finally, the positional embedding is encodes a word's position in that sentence. This embedding E is then fed into the network and the last hidden state of the [CLS] token can be seen as agregate sequence representation that can be used for classification (Devlin et al., 2019) as seen on Figure 2b.


Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}
(a)



(b)

Figure 2: (a) BERT's input representation and (b) BERT for single sequence classification (Devlin et al., 2019).

The distilBERT model is a case of transfer learning technique called distillation where a ”student” model is trained to achieve the behaviour of a larger ”teacher” model (Sanh et al., 2020). This managed to ”reduce the size of a BERT model by 40%, while retaining 97% of its language understanding capabilities and being 60% faster” (Sanh et al., 2020).

2.4 Linear Support Vector Classification

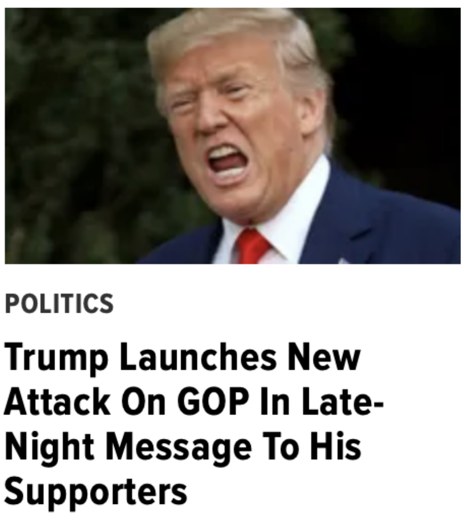
Given pairs of observations – word embeddings – and labels – sarcastic or not – $(\mathbf{x}_i, y_i)$, a support vector classifier tries to find the hyperplane which maximizes the margins between classes (Peña, 2019). The kernel used in classification can vary, but for NLP applications where the numbers of features are large, the linear kernel is more suitable given that text data is often linearly separable and the computations are faster (Hsu et al., 2003). The prediction is given by Equation 9 and the maximum margin is given by Equation 10 (Peña, 2019)

$$\hat{y} = \mathbf{x}\mathbf{w} + b \tag{9}$$

$$argmax_{\mathbf{w},b} \frac{1}{||\mathbf{w}||} \tag{10}$$

3 Data

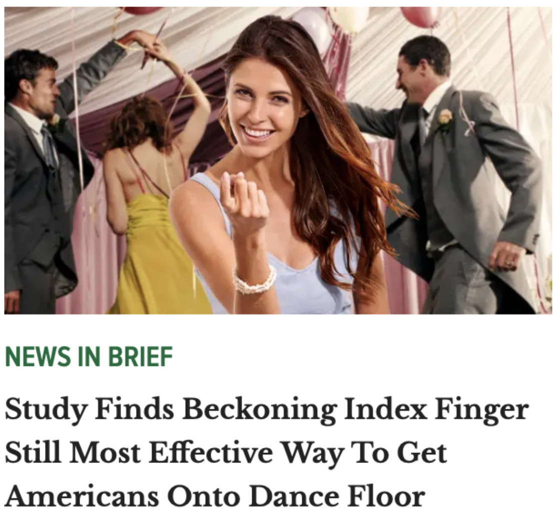
The data used was the News Headlines Dataset For Sarcasm Detection(Misra and Arora, 2019) found on Kaggle. This dataset consists of news headlines from two different websites: The Onion, a satiric news website where all headlines and articles are sarcastic/humorous, and Huffpost, a real news website where its headlines are considered 'not sarcastic'. It is clear that there is a difference in tone and language when it comes to these websites, as evidenced in Figure 3a and 3b.



POLITICS

Trump Launches New Attack On GOP In Late-Night Message To His Supporters

(a)



NEWS IN BRIEF

Study Finds Beckoning Index Finger Still Most Effective Way To Get Americans Onto Dance Floor

(b)

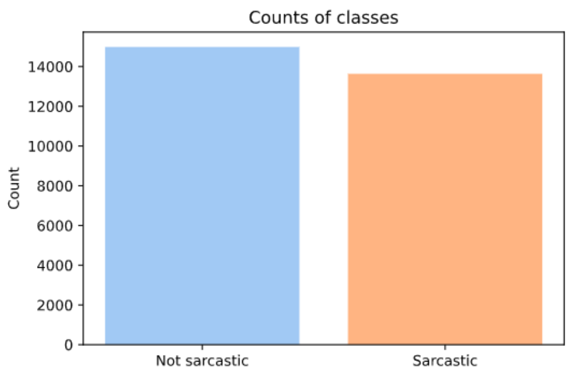
Figure 3: Sample of (a) not sarcastic and (b) sarcastic headlines from Huffpost and The Onion.

The raw data is in JSON format and contains information on 28619 headlines. The fields available are:

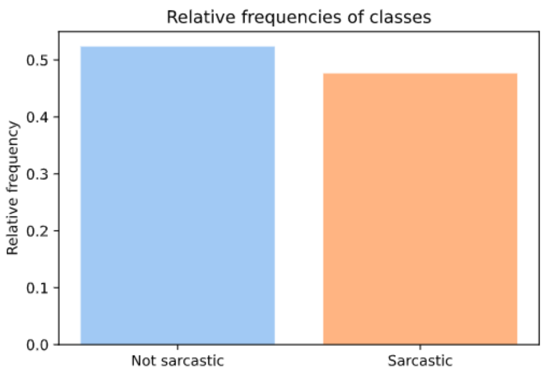
- is_sarcastic: Boolean.Wheter the headline is sarcastic or not.
- headline: String. Headline text.
- article_link: String. URL pointing to the headline's article.

The analysis of this report is focused on the headline text and its class. The article link, therefore, is discarded. Apart from that, no further preprocessing was done.

The classes were found to be fairly balanced, as shown in Figure 4a and 4b: 14985 non-sarcastic headlines against 13634 sarcastic ones, that is 52,36% and 47,64% of the total headlines respectively. A sample of the dataset is available in Table 1.



(a)



(b)

Figure 4: (a) Absolute count and (b) Relative frequencies of classes.

Text	Label
'thirtysomething scientists unveil doomsday clock of hair loss'	1
'dem rep. totally nails why congress is falling short on gender, racial equality'	0
'eat your veggies: 9 deliciously different recipes'	0
'inclement weather prevents liar from getting to work'	1
"mother comes pretty close to using word 'streaming' correctly"	1
'my white inheritance'	0
'5 ways to file your taxes with less stress'	0
"richard branson's global-warming donation nearly as much as cost of failed balloon trips"	1

Table 1: Sample of final dataset.

4 Method

In this section there will be details on how the study was carried, such as packages, pre-trained models and functions used. For all methods, the same training and test sets were used: 67% of headlines were used for training and 33% for testing. A fixed random seed of 42 ensures the reproducibility of results throughout the different methods.

Stop word removal and lemmatization were initially tested, but yielded worse results than the raw text. For this reason, no special pre-processing of text was used.

All methods are made of some type of vectorization of text and then fed to a `LinearSVC` classifier. Accuracy, recall and F1 scores were computed using Scikit-Learn's `classification_report`. Additionally, the `plot_confusion_matrix` was used to visualize the results. For even more details on coding, the reader is referred to Section 8 - Appendix.

4.1 Baseline: Random chance

The use of Scikit-Learn's `DummyClassifier` and `Pipeline` tools made this evaluation straightforward. The classifier was configured to the default stratified method, i.e. it will respect the class distribution (as stated previously, 52,36% not sarcastic and 47,64% sarcastic). After the instantiation of the pipeline, it was just a matter of fitting the classifier to training data and predicting on test data via the methods `pipeline.fit` and `pipeline.predict`.

4.2 TF-IDF

The key tool used in this section was Scikit-Learn's `TfidfVectorizer` function inside the pipeline. The `max_features` and `ngram_range` were set to 1000 and (1,2), respectively. The first parameter will only retrieve the top 1000 words in term frequency, while the second parameter will extract unigrams and bigrams from the corpus. After that, the classification routine proceeds as usual.

4.3 Word2Vec - CBOW

To extract Word2Vec embeddings, the spaCy library was used. The pre-trained pipeline `en_core_web_lg` comes with built-in, 300 dimensional, word vectors. Every document (i.e. headline) was vectorized as the average of their token vectors (Honnibal et al., 2020) and fed to the classifier.

4.4 DistilBERT

Pre-trained tokenizer and model were used in this section. More specifically, Hugging Face's `distilbert-base-uncased` model. The steps involved were adapted from Alammar (2019).

The tokenizer step preparess text to be fed to the distilBERT network. More importantly, it adds the special [CLS] (which has an id of [101]) token at the beginning of every sequence, which will be crucial to classification later on. Given the average word count in all headlines was around 10 words/headline and that the vast majority of headlines were 12 words or shorter as seen on Table 2, an embedding dimension of 25 was chosen as a good compromise between shorter and longer headlines. That way, shorter headlines will be padded with zeroes and longer headlines will be truncated. distilBERT's tokenizer not only tokenize words, but it will also split words into pieces (wordpiece tokenization).

Statistics	Value
Headlines	28619
Average words per headline	10.05
Standard deviation	3.39
Min	2
25%	8
50%	10
75%	12
Max	151

Table 2: Descriptive statistics of sentences.

An example of tokenization is as follows:

——A raw text headline: `thirtysomething scientists unveil doomsday clock of hair loss`

—— This is the (wordpiece) tokenized headline, padded: `['[CLS]', 'thirty', '##some', '##thing', 'scientists', 'un', '##ve', '##il', 'doom', '##sd', '##ay', 'clock', 'of', 'hair', 'loss', '[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]']`

——This is the token ID headline, padded: `[101, 4228, 14045, 20744, 6529, 4895, 3726, 4014, 12677, 16150, 4710, 5119, 1997, 2606, 3279, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0]`

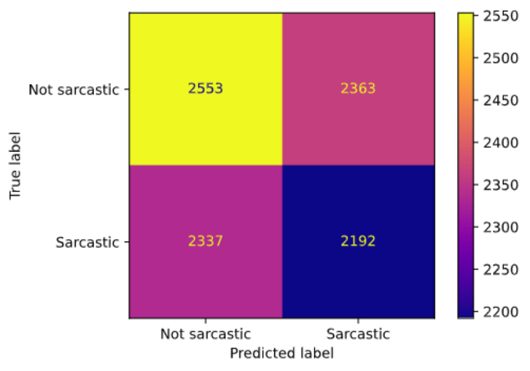
The token id vectors were then fed to the distilBERT network. For every input (token id), 768 hidden units were outputted. The only hidden units of interest here are the ones tied to the [CLS] token. After slicing the output tensor to retrieve those units, they were then fed to the classifier.

5 Results

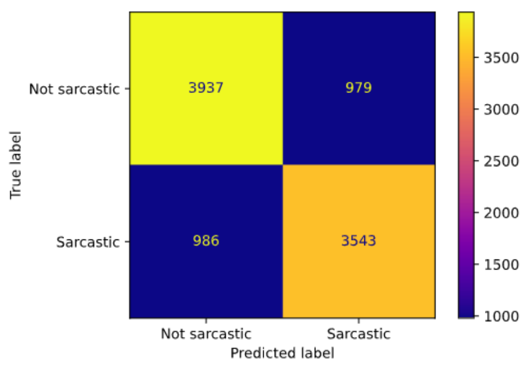
In addition to classification accuracy, relative increase of this metric with respect to the previous one is displayed on Table 3. Random choice, as expected, performed worse with 50% accuracy, followed by a significant increase with TF-IDF vectorization. CBOW embeddings slightly improved accuracy up to 81%. DistilBERT embeddings achieve the highest accuracy, 86%, a 6.2% increase from CBOW and a 72% increase relative to the dummy classifier. These results are also reflected in Figure 5 where the confusion matrices for all methods are displayed.

Method	Accuracy	Relative increase
Random choice	0.5	0%
TF-IDF	0.79	58%
CBOW	0.81	2.5%
distilBERT	0.86	6.2%

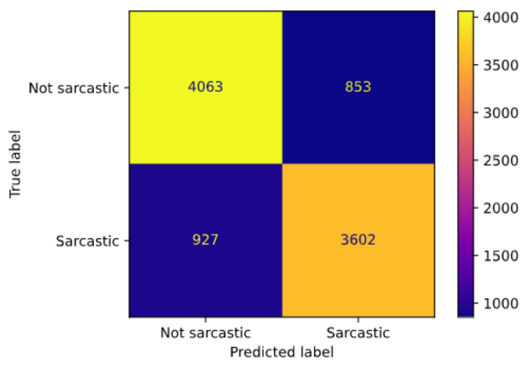
Table 3: Accuracy for different vectorization/embeddings.



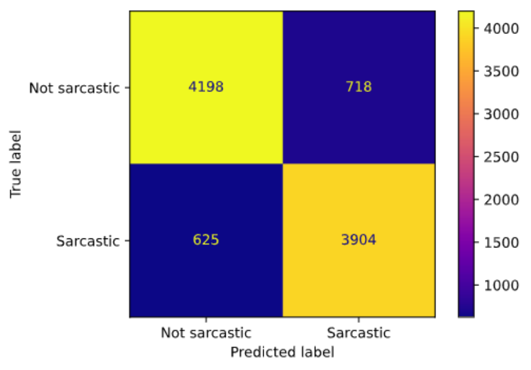
(a) Random choice



(b) TF-IDF



(c) CBOW



(d) distilBERT

Figure 5: Confusion matrices

6 Discussion

From the results of the previous section, it is evident that more powerful algorithms/models yield best classification results. This is specially the case for distilBERT, who achieved the highest accuracy of all tested methods. This can be explained by context capturing (via positional embeddings), which distilBERT does while the other methods do not. Still, CBOW achieved relatively good metrics. On the other hand, distilBERT was significantly more computationally heavy: this is considered a small dataset and even with a small 25 tokenized dimensions, the evaluation took almost an hour to train. Of course, the use of parallelization (which is highly recommended) was not used, which in itself could be an improvement for next projects/larger corpora.

One clear limitation of the work is in the data itself. The fact that a headline comes from a ”serious” news website does not imply the absence of sarcasm. Sarcasm is a good communication tool to transmit opinion and is wildly used throughout media, serious or not. As an improvement, more traditional sarcasm datasets could be used to compare with previous work, such as the Twitter and/or Reddit corpus.

Another limitation was the somewhat simple conduction of experiments: a lot more could have been done in terms of hyperparameter tuning, both for the vectorization/embeddings and for the classifier. Some examples of this could be vocabulary size in TF-IDF, different tokenization dimensions, and number hidden layers/attention heads in distilBERT. Also other classifiers could have been used for comparison.

The biggest limitation, however, was that the distilBERT model was not fine-tuned for this task. It is believed the even better accuracy could have been achieved with it. Nonetheless, the higher performance of transformer models in figurative language detection is not surprising, as evidenced in (Potamias et al., 2020).

7 Conclusion

While this project had its limitations, it was possible to see that indeed transformer networks such as distilBERT are more suitable for highly context-dependent NLP tasks such as sarcasm detection. The power of transfomers comes with a caveat: these models are significantly more computationally demanding and need to use parallelization to be efficient. The results could be improved via a more optimized dataset and fine-tuning.

In conclusion, a not-so-black-box view of this project allowed for an initial contact with the transformer architecture, its power, and its importance to the NLP world.

8 Appendix: Code and data

The preprocessed data and code used in the making of this report can be found in this repository.

References

Alammar, J. (2019). A visual guide to using bert for the first time. *Jay Alammar's personal Blog.*

Devlin, J. (2019). Contextual word representations with bert and other pre-trained language models. Powerpoint presentation. Standford University.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

Ethayarajh, K. (2020). Bert, elmo, gpt-2: How contextual are contextualized word representations? *The Stanford AI Lab Blog.*

Ghodsi, A. (2017a). Ali ghodsi, lec 12: Neural networks, autoencoders, word2vec. `https://youtu.be/syWB-YMYZvI`.

Ghodsi, A. (2017b). Ali ghodsi, lec 13: Word2vec skip-gram. `https://youtu.be/GMCwS7tS5ZM`.

González-Ibánez, R., Muresan, S., and Wacholder, N. (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.

Halthor, A. (2020). Bert neural network - explained! `https://www.youtube.com/watch?v=xI0HHN5XKDo`.

Honnibal, M., Montani, I., Van Landeghem, S., and Boyd, A. (2020). spaCy: Industrial-strength Natural Language Processing in Python.

Hsu, C.-W., Chang, C.-C., Lin, C.-J., et al. (2003). A practical guide to support vector classification.

Kuhlmann, M. (2020). Text classification. 732A92/TDDE16 Text Mining lecture slides, Linköping University.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval.* Cambridge University Press, Cambridge, UK.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

Misra, R. and Arora, P. (2019). Sarcasm detection using hybrid neural network. *arXiv preprint arXiv:1908.07414.*

Patro, J., Bansal, S., and Mukherjee, A. (2019). A deep-learning framework to detect sarcasm targets. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6337–6343.

Peña, J. M. (2019). Lecture 3b block 1: Support vector machines. 732A99/TDDE01 Machine Learning lecture slides, Linköping University.

Potamias, R. A., Siolas, G., and Stafylopatis, A. G. (2020). A transformer-based approach to irony and sarcasm detection. *Neural Computing and Applications*, 32(23):17309–17320.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2020). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.