Final Assignment

How do text pre-processing and Machine Learning algorithms affect sentiment analysis? A research on Twitter

Student ID: 2022103816

January 10, 2023

Abstract

In this work, we focus on the sentiment analysis of Tweets (i.e. messages posted on Twitter), and analyze how text pre-processing techniques and different machine learning algorithms affect the accuracy of sentiment analysis. What's noteworthy is that we propose a novel measure called $NFDF$ to identify important words, in order to construct the vocabulary for a Naive Bayes classifier, and write a Python Class in which Naive Bayes is implemented from scratch. Besides, for the sake of convenience, we integrate all the text pre-processing functions into a Python Class. We come to the following conclusions: (a) Lemmatization only helps when the vocabulary of Naive Bayes is sufficiently small, probably because it spares more space for words with different meanings. As the vocabulary size gets bigger, lemmatization begins to harm model performance. For LSTM and BERT, lemmatization never helps, probably because both of them take raw sequences of words as input and thereby are capable of learning sentence syntax, while lemmatization breaks syntax. (b) The deletion of stopwords never benefits model performance, and this indicates that stopwords might play an important role in the sentiment analysis of casually written texts(eg. tweets). But this still needs to be verified by checking the effect of stopword deletion in formal texts like news. (c) BERT model with one fully-connected output layer and without lemmatization or stopword deletion achieves the best predicting performance on validation set. This model eventually achieves an overall accuracy of 83.6% on test data, with recalls of positive and negative tweets equal to 80.2% and 87.0% respectively.

Keywords: sentiment analysis, naive bayes, lstm, bert, lemmatization, stopword

1 Introduction

Sentiment analysis, pioneered by Pang et al. (2002), is an important subdomain of Natural Language Processing, which is aimed at determining if a piece of text has a positive or negative sentiment. There are a wide range of scenarios where sentiment analysis could help, such as when a company is doing research on how customers think of their products, or when someone wants to capture investor sentiment in order to predict stock prices. What's more, with the rapid development of social media, billions of people post messages online every day. Some of these messages contain harmful content like racist or sexist speech, while some others may reflect depression or intention of suicide that should be noticed before someone gets into serious trouble. Therefore, in this work, we primarily focus on the sentiment analysis of tweets (i.e. messages posted on Twitter), try out various text pre-processing techniques and machine learning algorithms, and compare their performance. All the codes and datasets involved in this work can be found at: https://github.com/WoodySJR/Prog_Project.

2 Data Description and Exploratory Analysis

2.1 Data Description

In this work, we utilize the dataset called "Sentiment140" available at: http://help.sentiment140.com/for-students/, which started as a class project from Starford University in 2009. This dataset was collected through Twitter Search API by Go et al. (2009). The training set of size 1.6 million was automatically annotated as positive or negative, depending on whether the tweets include positive or negative emoticons, such as :) and :(; The test set of size 359 was manually labelled. Due to the limitation of computational resources, we only keep 20,000 positive and 20,000 negative comments from the training set, and further extract 20% to be our validation set for hyperparameter tuning and model selection. We finally evaluate the predicting accuracy of our model on the test set.

2.2 Exploratory Analysis

The average number of words in positive tweets is 13.98, while that of negative tweets is 14.78, without much difference. 55.2% of positive tweets contain "@username", while it is only 38.2% for negative tweets, which indicates that positive tweets are more often directed to somebody. 5.8% of positive tweets contain weblinks, while for negative tweets it is only 3%. In order to get a glimpse of what positive and negative tweets look like, some examples are listed in Table 1, and we might as well draw separate wordclouds for them(as shown in Figure 1). It's easy to see that in positive tweets, words like "thank", "love" and "good" appears frequently, while in negative tweets, words like "work", "miss" and "want" are more frequent. Before modelling, the texts need to be pre-processed, which we will introduce in detail in Section 3.1.

Table 1: Examples of positive and negative tweets

Sentiment	Text
positive	Thanks for pointing out the crucial problems @thakkar.Both of them have been taken care of (cc: @Netra)
positive	Just observed the update look of #Alexa website, http://tinyurl.com/c48gzf , looks great, seems Alexa is improving
positive	@comeagainjen http://twitpic.com/2y2lx - and you are cooler than her she just wants the babiiesss O_O HAHAHAHA
negative	my whole body feels itchy and like its on fire
negative	@nationwideclass no, it's not behaving at all. i'm mad. why am i here? because I can't see you all over there.
negative	@caregiving I couldn't bear to watch it. And I thought the UA loss was embarrassing . . . . .

(a) Positive tweets

(b) Negative tweets

Figure 1: Wordclouds

3 Methods

In this section, we elaborate on the methods adopted in this work.

3.1 Text pre-processing techniques

There are certain unique properties of tweets, compared with other kinds of texts in sentiment analysis. For one, tweets contain very casual language, such as misspellings, slang, and repeated letters for emphasis (eg. awesoooome); For another, tweets very often contain @usernames and URLs, which may contain more noise than signal. Therefore, we pre-process the tweets in the following ways.

- We replace all "@username" with the same token "[USERNAME]", and replace all URLs with "[URL]". The reason why we do not simply delete them is because their existence might still provide useful information, as pointed out in Section 2.2.
- We replace any letter that occurs more that twice in a row with two occurrences (eg. awesoooome → awesoome);

- For traditional text representation (like Bag Of Words), it's common practice to drop all stopwords (i.e. words with no specific meanings, like "a", "the" and "is"). Besides, the remaining words are often stemmed or lemmatized (eg. ate $\rightarrow$ eat) to reduce feature space[1]. However, such transformations break the original sentence structure. As deep learning algorithms begin to catch on in NLP, which directly take sequences of words as inputs, we may wonder whether these pre-processing techniques are still beneficial. Therefore, one of our objectives in this work is to check whether **stopwords deletion** and **word lemmatization** improve model performance.

- Besides, we convert all letters to their lower forms, and delete all digits and punctuations.

Note: For the sake of convenience, we wrote a Python class named **preprocess** to integrate all these functions. What's worth noting is that after pre-processing, some texts become empty(i.e. nan), which belong to the 'float' object. However, during model training, we need to split each text into tokens using str.split() function, and when it comes to nan we will have "AttributeError". Therefore, we handle the error with "try...except...", and skip those empty texts when "AttributeError" is encountered. Besides, since "lemmatization" and "stemming" serve similar purposes, we raise a warning message when both of them is activated. Demonstrated in Table 2 are a couple of tweets before and after pre-processing.

Table 2: Tweets before and after pre-processing

Index	Before	After
1	ok I'm sick and spent an hour sitting in the shower cause I was too sick to stand and held back the puke like a champ. BED now	ok i'm sick spend hour sit shower cause sick stand hold back puke like champ bed
2	mornnnninggg. ugh by cub has gone to work without a phoneee. got no one to textt @SamShepherd	mornningg ugh cub go work without phonee get one textt USERNAME
3	319 more followers til 1k not happenin anytime soon http://PetParenthood.blogspot.com	follower til k happenin anytime soon URL

3.2 Machine learning algorithms

In this work, we try out three machine learning algorithms for sentiment analysis, namely Naive Bayes, LSTM and BERT, and compare their performances.

3.2.1 Naive Bayes Classifier

Let us represent each piece of text as a bag of words $\{w_1, w_2, \cdots, w_n\}$, where repetition is allowed, and denote its sentiment as y =1 or 0, depending on whether it is positive or negative. By Bayes' rule, the posterior probability of a piece of text being positive, given its bag of words, is:

$$\mathbb{P}(y=1|w_1,w_2,\cdots,w_n) = \frac{\mathbb{P}(y=1)\mathbb{P}(w_1,\cdots,w_n|y=1)}{\mathbb{P}(y=1)\mathbb{P}(w_1,\cdots,w_n|y=1)+\mathbb{P}(y=0)\mathbb{P}(w_1,\cdots,w_n|y=0)}.$$

[1]The difference between lemmatization and stemming is subtle. Stemming keeps the stem of a word in a heuristic way, sometimes resulting in meaningless stems(eg. happiness→happi). Lemmatization takes the context into consideration, and returns the base form of a word by looking in up in WordNet(eg. meeting(noun) is returned unchanged, while meeting(verb) is lemmatized into meet). In this work, we focus on lemmatization.

To simplify calculation, we further assume conditional independence among w_i, given $y = 1$ or 0, hence:

$$\mathbb{P}(y=1|w_1,\cdots,w_n) = \frac{\mathbb{P}(y=1)\prod_{i=1}^n \mathbb{P}(w_i|y=1)}{\mathbb{P}(y=1)\prod_{i=1}^n \mathbb{P}(w_i|y=1) + \mathbb{P}(y=0)\prod_{i=1}^n \mathbb{P}(w_i|y=0)}. \tag{3.1}$$

We estimate $\mathbb{P}(w_i|y=1)$ with the frequency of w_i in positive tweets, i.e.,

$$\hat{\mathbb{P}}(w_i|y=1) = \frac{\#\text{appearance of } w_i \text{ in positive tweets}}{\#\text{words in all positive tweets}},$$

and estimate $\mathbb{P}(y=1)$ with the frequency of positive tweets among all tweets, i.e.,

$$\hat{\mathbb{P}}(y=1) = \frac{\#\text{positive tweets}}{\#\text{positive tweets} + \#\text{negative tweets}}.$$

The estimation of $\mathbb{P}(w_i|y=0)$ and $\mathbb{P}(y=0)$ follows the same way. Plugging these estimates into (3.1), we obtain an estimated probability of a tweet being positive or negative, given its bag of words. A crucial problem in Naive Bayes is the construction of vocabulary. In Information Retrieval, TF-IDF(term frequency-inverse document frequency) is utilized to identify important words in distinguishing different documents. If a word w_i appears frequently in a document D_j, and meanwhile it only appears in a small proportion of documents, then it is deemed to be important for this very document. In this work, we propose a modified version of TF-IDF to suit our purpose. For a specific word w_i, first calculate its frequencies in positive and negative tweets respectively, which are basically $\hat{\mathbb{P}}(w_i|y=1)$ and $\hat{\mathbb{P}}(w_i|y=0)$ as mentioned above. Then, compute the following measure:

$$\frac{\Big||\{d \in \mathcal{D}_+ : w_i \in d\}| - |\{d \in \mathcal{D}_- : w_i \in d\}|\Big|}{\max\{|\mathcal{D}_+|, |\mathcal{D}_-|\}}, \tag{3.2}$$

where $\mathcal{D}_+$ and $\mathcal{D}_-$ denote the sets of positive and negative tweets respectively, and the denominator serves as a normalizing constant. This measure reflects the difference of w_i's frequencies in positive and negative tweets, and we shall denote it as DF_i, which stands for **d**ifference in **f**requency. Finally, we obtain an importance measure of w_i for positive tweets by multiplying $\mathbb{P}(w_i|y=1)$ and DF_i together, and denote it as $TFDF_i^+$. Furthermore, we take d words with the biggest $TFDF_i^+$, and denote them as $\mathcal{V}^+$. Similarly, we obtain $\mathcal{V}^-$, and we pool $\mathcal{V}^+$ and $\mathcal{V}^-$ together (with duplicated words discarded) to be our vocabulary. Here, d is a tuning parameter controlling the size of our vocabulary.

Note: We implement Naive Bayes from scratch, and write a class named **naive_bayes** to integrate all its functions, including the calculation of $TFDF$, construction of vocabulary, parameter estimation, and prediction.

3.2.2 LSTM

Long Short-Term Memory(LSTM)(Hochreiter and Schmidhuber, 1997) is one of the variants of recurrent neural network(RNN), and it is proposed to tackle the problem of "gradient vanishing" with the original RNN. In simple terms, in the original RNN, the encoding of a specific word would be dominated by its immediate neighbors, and words at a distance would be easily forgotten. Therefore, LSTM introduces into RNN a "cell state", which serves as long-term memory, in addition to the "hidden state" in original RNN which serves as short-term memory. Besides, LSTM utilizes "gates" to control the information flow between short-term and long-term memory. Thus, LSTM is better at capturing long-term dependencies in a piece of text.

3.2.3 BERT

Bidirectional Encoder Representation from Transformers(BERT)(Devlin et al., 2018) utilizes the self-attention mechanism in transformers(Vaswani et al., 2017) for text representation, and innovatively proposes pre-training on large unlabelled corpora to acquire prior knowledge. The pre-training involves two tasks: one is masked language model, which is to predict a word based on its context; the other is next sentence prediction, which is to predict whether two arbitrary sentences are adjacent in a document. The model is then fine-tuned on a handful of domain-specific labelled examples, and would be qualified for a variety of downstream tasks. In this work, we employ the pre-trained BERT model called "book_corpus_wiki_en_uncased", which is pretrained by Google on English Wikipedia and books, and fine-tune its parameters on our dataset.

Note: The mathematical details of LSTM and BERT are omitted due to lack of space. We build LSTM and BERT models under the deep learning framework "MXNET".

4 Experiments

In this section, we present our implementation details, evaluation metrics of model performance, and experimental results.

4.1 Implementation details

- For Naive Bayes, we tune the vocabulary size d as mentioned in Section 3.2.1 as a hyperparameter, with optional values $\{100, 500, 1000, 2000, 5000\}$. Laplace smoothing is adopted in calculating $\hat{\mathbb{P}}(w_i|y=1)$ and $\mathbb{P}(w_i|y=0)$ to avoid zero probability. That is, the equation (3.2) is modified into:

$$\hat{\mathbb{P}}(w_i|y=1) = \frac{1 + \#\text{appearance of } w_i \text{ in positive tweets}}{\#\text{vocabulary} + \#\text{words in all positive tweets}}, \tag{4.1}$$

  and $\hat{\mathbb{P}}(w_i|y=0)$ is modified in a similar way.

- For LSTM, the number of hidden units is tuned among $\{64, 128, 256\}$, the number of LSTM layers is tuned between $\{1, 2\}$, and the number of fully-connected output layers is also tuned between $\{1, 2\}$. The 300-dimensional pretrained word embeddings "GloVe"(Pennington et al., 2014) are employed in word representation.

- For BERT, since it is pretrained with prespecified model architecture, we don't have many options for its hyperparameters. Therefore, we only tune the number of fully-connected output layers between $\{1, 2\}$.

- Moreover, for each set of hyperparameter, we try four ways of text pre-processing, namely (1)lemmatization and stopword deletion, (2)lemmatization, (3)stopword deletion and (4) neither, in order to see how text-preprocessing affect sentiment analysis.

- The learning rates of LSTM and BERT are set to be 1×10^{-3} and 1×10^{-4} respectively. Xavier uniform(Glorot and Bengio, 2010) is utilized wherever random initialization is needed. The Adam optimizer(Kingma and Ba, 2014) is used in model training with momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Each model is trained for 10 epochs and evaluated after each epoch on the validation set. The best performance is presented as our experimental results. For more implementation details, please refer to the codes on Github.

4.2 Evaluation metrics

We adopt AUC (Area under the ROC curve) as the evaluation metric in hyperparameter tuning and model selection. We evaluate our best model on the test set in terms of AUC, recalls of positive and negative tweets, and the overall accuracy. The probability threshold in calculating recalls is determined by maximizing the Youden Index(Youden, 1950).

4.3 Experimental results

4.3.1 Naive Bayes

As can be seen from Figure 2, for a vocabulary of size 100, lemmatization improves predicting performance, probably because it spares more space for words with different meanings. However, with the vocabulary size increased to 500, the benefit of lemmatization becomes less obvious. As the vocabulary size exceeds 1000, lemmatization begins to harm model performance. It can also be seen that the deletion of stopwords always harms model performance, which indicates that stopwords play an important role in the sentiment analysis of more casually written texts (eg. Tweets). Model performance is optimized(AUC=0.835) with a vocabulary of size 5000, and without lemmatization or stopword deletion.

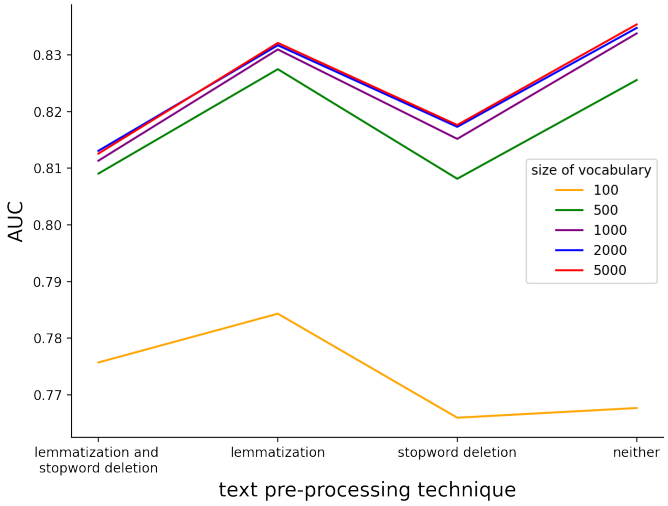
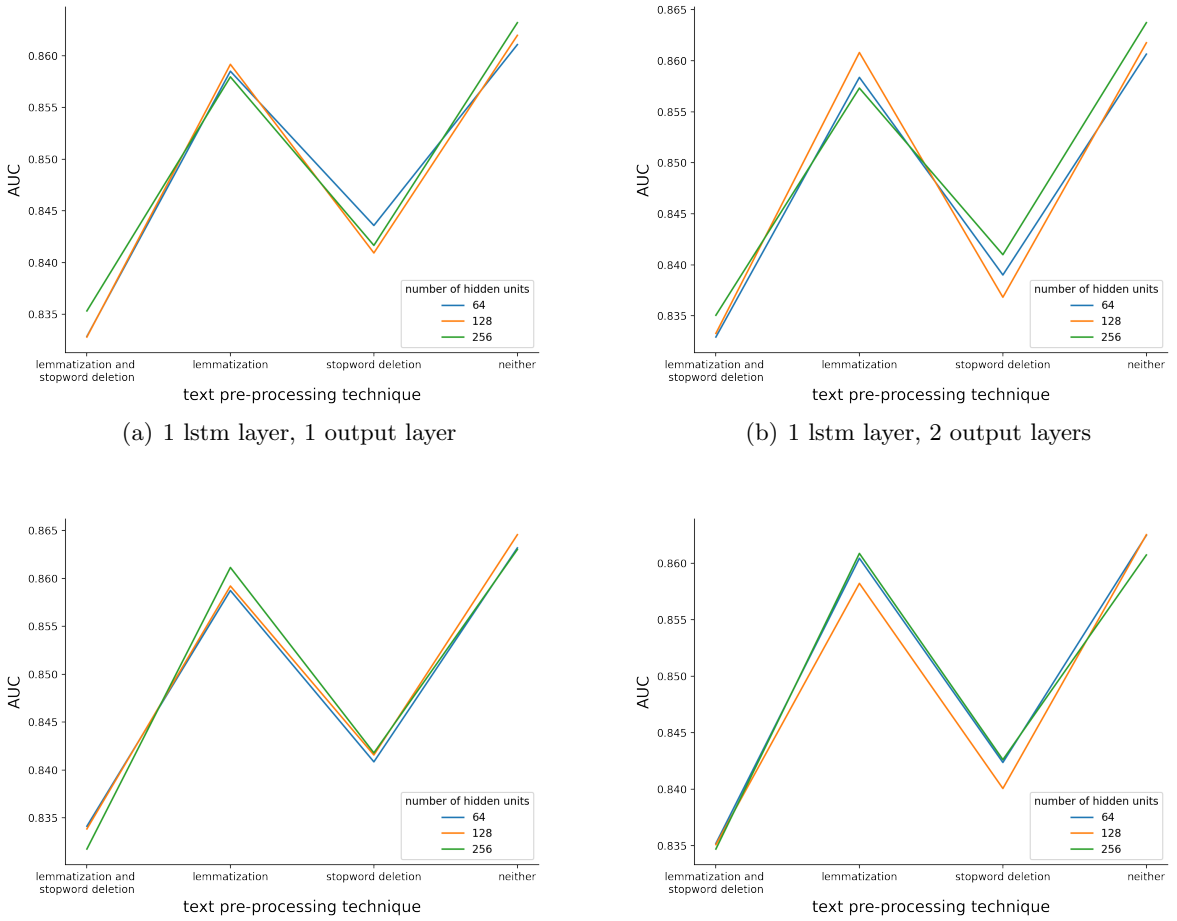


Figure 2: Hyperparameter tuning of Naive Bayes

4.3.2 LSTM

As can be seen from Figure 3, LSTM performs consistently better than Naive Bayes, without much doubt. This is because LSTM is specifically designed for sequential data, and capable of learning long-term dependencies. For LSTM, both lemmatization and stopword deletion harm model performance. This is probably because that, LSTM directly takes sequences of words as input and is thereby capable of learning about syntax(i.e. the grammatical arrangement of words) in sentences. However, lemmatization and stopword deletion both break the original syntax, which is bad news for LSTM. The performance of LSTM is optimized(AUC=0.865) with 128 hidden units, two LSTM layers and one output layer, and without lemmatization or stopword deletion.



(a) 1 lstm layer, 1 output layer

(b) 1 lstm layer, 2 output layers

(c) 2 lstm layers, 1 output layer

(d) 2 lstm layers, 2 output layers

Figure 3: Hyperparameter tuning of LSTM

4.3.3 BERT

As can be seen from Figure 4, lemmatization and stopword deletion both harm the performance of BERT, probably for the same reason as LSTM. What's noteworthy is that, with lemmatization and stopword deletion, BERT even performs more poorly than LSTM. We suppose one of the reasons is that, BERT is already pretrained on large corpora whose complete syntax is maintained. Therefore, BERT is more accustomed to natural language than LSTM is, and the damage of sentence structure would go against its prior knowledge, making it lose its edge. The performance of BERT is optimized(AUC=0.873) with one output layer and neither lemmatization nor stopword deletion.

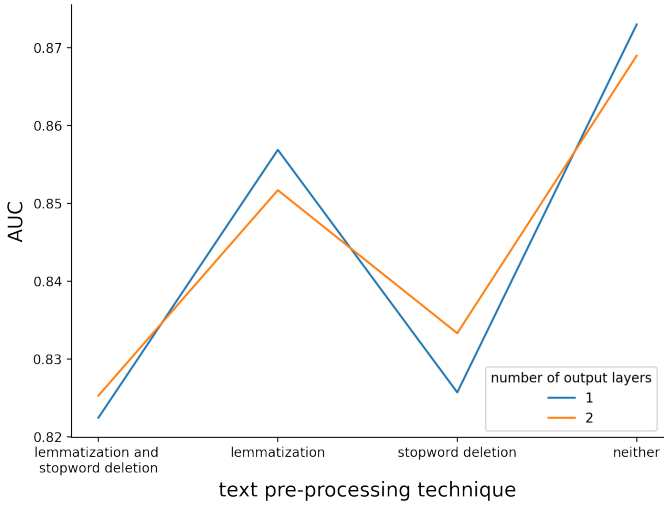


Figure 4: Hyperparameter tuning of BERT

4.3.4 Final evaluation

Based on the above results, a BERT model with one output layer and without lemmatization or stopword deletion would be our best choice, which verifies the power of large pretrained neural networks. We re-train this model on our whole training set(i.e. with the validation set included), and evaluate its performance on the test set. The results are listed in Table 3, which are quite satisfying.

Table 3: Final evaluation of model performance

Metric	Value
AUC	0.889
Best probability threshold	0.579
Recall of positive tweets	80.2%
Recall of negative tweets	87.0%
Overall accuracy	83.6%

5 Conclusions

To summarize, in this work, we focus on the sentiment analysis of Tweets (i.e. messages posted on Twitter), and analyze how text pre-processing techniques and different machine learning algorithms affect the accuracy of sentiment analysis. What's noteworthy is that we propose a novel measure called $NFDF$ to identify important words, in order to construct the vocabulary for a Naive Bayes classifier, and write a Python Class in which Naive Bayes is implemented from scratch. Besides, for the sake of convenience, we integrate all the text pre-processing functions into a Python Class. We come to the following conclusions:

- Lemmatization only helps when the vocabulary of Naive Bayes is sufficiently small, probably because it spares more space for words with different meanings. As the vocabulary size gets

bigger, lemmatization begins to harm model performance. For LSTM and BERT, lemmatization never helps, probably because both of them take raw sequences of words as input and thereby are capable of learning sentence syntax, while lemmatization breaks syntax.

- The deletion of stopwords never benefits model performance, and this indicates that stopwords might play an important role in the sentiment analysis of casually written texts(eg. tweets). But this still needs to be verified by checking the effect of stopword deletion in formal texts like news.
- BERT model with one fully-connected output layer and without lemmatization or stopword deletion achieves the best predicting performance on validation set. This model eventually achieves an overall accuracy of 83.6% on test data, with recalls of positive and negative tweets equal to 80.2% and 87.0% respectively.

For future work, more machine learning algorithms could be tried, and the pre-processing of tweets could be further refined, such as correcting the misspellings. We could also divide sentiment into more precise categories, like excitement, hatred and sadness, so as to enable more applications.

Due to the author's limited knowledge, there must be some mistakes in this work. Your criticisms and suggestions would be greatly appreciated. Thank you for your time!

References

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Go, A., Bhayani, R., and Huang, L. (2009). Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 1(12):2009.

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

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Pang, B., Lee, L., and Vaithyanathan, S. (2002). Thumbs up? sentiment classification using machine learning techniques. *arXiv preprint cs/0205070*.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Youden, W. J. (1950). Index for rating diagnostic tests. *Cancer*, 3(1):32–35.