Twitter Sentiment Analysis

A survey of NLP and sentiment analsyis

Elijah Bayani, Reed Bertolotti, Halleh Bostanchi, Adam Chen,
Seth Owings, Daniel Zavorotny

June 4, 2022

Contents

1 Introduction and Problem Statement

Natural language processing (NLP) is a combination of computational linguistics and machine learning that allows computers to process and interpret human languages through software (Catania, 2021, p. 51). The human language being processed through NLP may come in various forms, such as social media posts, product/service reviews, or survey responses. NLP algorithms are used on the language inputs to convert it into code a computer can understand using syntactic and semantic rules that matches the language of the input (Catania, 2021, p. 51). NLP is used in a plethora of applications, such as translating language, grammar correction software, or digital person assistants such as Siri (Catania, 2021, p. 52).

To understand sentiment analysis, we must first define sentiment. Sentiment is the expression of subjectivity through language, which may be characterized by positive or negative evaluation (Taboada, 2016, p. 326). Accordingly, sentiment analysis is a combination of computer science and linguistics that tries to automatically identify sentiment

within a text (Taboada, 2016, p. 325). Utilizing machine learning techniques is an option to perform sentiment analysis, which generally consists of using a classifier created through supervised learning by being trained on sentiment-labeled data (Taboada, 2016, p. 327). The classifier then learns to differentiate positive tokens from negative tokens which are then used to determine the sentiment of the entire text. Within machine learning used for sentiment analysis, there are two further approaches: the linear approach, including Support Vector Machines or Artificial Neural Networks; or the deep learning approach, including Deep Neural Networks or Convolutional Neural Networks (Birjali, Kasri, & Beni-Hssane, 2021, p. 9)).

The objective of this project is to apply sentiment analysis to Twitter. Specifically, when given a tweet, it aims to accurately classify its sentiment. In a larger sense, this project is also a survey of various NLP and sentiment analysis models and techniques.

2 Background Research and Literature Review

Sentiment analysis is useful for predicting customer trends and public opinions by gaining insight on positive and negative sentiments gathered from relevant text sources (Gupta, 2018). In general, sentiment analysis allows businesses to take advantage of unstructured data to gain insights that may be acted on for company benefit. As a result, businesses are now able to determine the sentiment of a company's or brand's demographic pertaining to certain services, procedures, products, or policies and adjust to meet the demographics needs or utilize more appealing branding techniques (Gupta, 2018). This provides an enormous benefit to businesses as it gives them a way of better understanding their customers' opinions to improve their products/services or change marketing strategies, especially in situations where the quantity of data is too large to be manually processed. Sentiment analysis may also be utilized in various other ways such as to improve healthcare from reviews of patients or determining public opinions regarding politics or social issues for government intelligence (Birjali et al., 2021, p. 4).

Our project drew inspiration from several sources. One such source was the GloVe, global vectors for word representation, project by the Stanford NLP group (Pennington, Socher, & Manning, 2014). As is discussed later with regards to word embedding, GloVe is a machine learning algorithm that turns words into vectors. It is useful in sentiment analysis and other NLP tasks.

Additionally, we researched top sentiment analysis approaches on Kaggle, an online data science competition community. The kernels or notebooks we looked at gave us ideas of which libraries would be useful for our sentiment analysis project. One notebook discussed the bag of words and TF-IDF approaches (Periwal, 2021). The other introduced us to LSTM, which is also discussed later (Pandian, 2020).

3 Exploratory Data Analysis

The dataset used in this project was obtained from a past Kaggle competition. Each example in the dataset consists of the text of a tweet, some selected text of the tweet, the tweet's unique ID, and the sentiment of the tweet. Tweet examples are labeled as positive,

negative, or neutral. The dataset contains about 27,000 examples, approximately evenly distributed among the three sentiments.

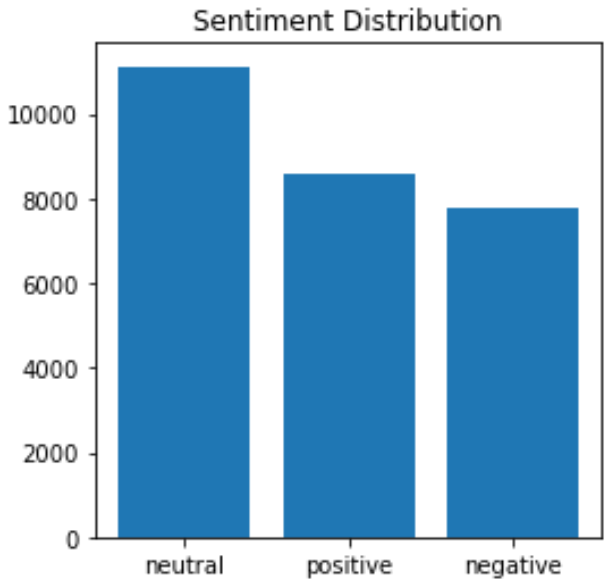


Figure 1: Distribution of dataset by sentiment class

The text of the tweet seems to not include emojis, which are popular on social media like twitter. Other than the absence of emojis, the text feature appears unprocessed. It is unclear how the labels were determined on the data, but some confidence can be had in their accuracy since the dataset was provided by the Kaggle organization in an official contest.

This project uses only the text feature of examples to predict the sentiment label. The other features were ignored. Tweet ID would not be helpful in predicting sentiment and selected text is just a subset of the total text.

4 Machine Learning Process

4.1 Baseline Models

Sentiment analysis on tweets is a nontrivial problem well suited to machine learning. We assumed that a complex model and hyperparameter tuning would be needed to get a high accuracy. So before spending lots of time formulating this model, training, and tuning, we decided to establish a baseline. We utilized a few simple models that are fast to create and train to establish a minimum acceptable accuracy.

Preprocessing

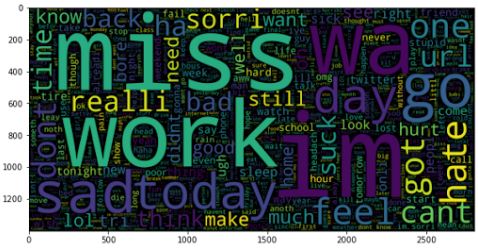
The ways to represent the text feature of a tweet rely on breaking it into words or tokens first. These words may be actual words or not. This preprocessing is done by removing

punctuation and splitting words on whitespace.

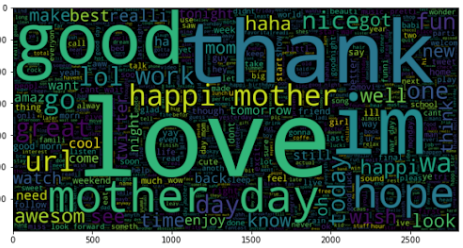
Additional preprocessing was needed to make the text more general and have less variation. As a result, words were lowercased and URLs were replaced with the word "URL." Similarly, for Twitter at (@) mentions, these were replaced with "USER." Furthermore, words that ended with three or more of the same character (e.g. noooo) were replaced with the word ending in two of its terminating character.

In the same vein of removing wide variation, some additional techniques from NLP were applied. Stop words, common words with little meaning like "the," were removed completely. Stemming was applied to words to get their base form (e.g. greater becomes great). Lemmatization was also used to transform words into their lemmas or general meaning base words (e.g. great becomes good).

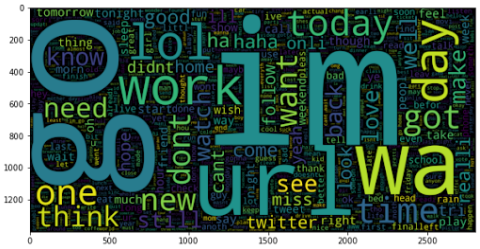
Now that the text feature can be split into words, some additional data visualization can be done. Word clouds can be generated for each class to visualize the most frequent words of each sentiment type. The idea to create word clouds came from the work of Periwal (2021) and Pandian (2020).



(a) Word cloud of tokens in negative sentiment tweets



(b) Word cloud of tokens in positive sentiment tweets



(c) Word cloud of tokens in neutral sentiment tweets

Figure 2: Word clouds

To represent the text of a tweet in a way the computer can understand, a bag of words model was used. A vocabulary was created from all words in the training set. Each word has an index in the vocabulary. The text feature of a tweet can be represented as the indices for its words in the vocabulary. A tweet is represented by a feature vector with the same size as the vocabulary. If the tweet has a certain word, the number of times that word occurs is stored in the feature vector at the index of the word in the vocabulary. If a word is not present in a tweet, a zero is stored to represent zero occurrences.

Training and Evaluation

Each of the baseline models was trained on the transformed text feature vectors of the training set. K-fold cross validation was also applied. The average was taken of the accuracies for each fold. The multinomial naive bayes, linear SVM, and logistic regression achieved 65.4%, 66.6%, and 69.5% average accuracy respectively. This established our baseline accuracy that we wanted to beat.

Feature Engineering

Towards the end of our project, we returned to the preprocessing step of the baseline models. We designed several new features using the properties of the preprocessed text for an example. Punctuation to total characters ratio, number of words, and average word length were some of the features, to name a few.

By this point in the project, the deadline was close at hand, so we did not get a chance to integrate the engineered features in our models. Some of them may have improved accuracy, but we believe most would have had no effect. We are confident that most would not have aided our models since the average for most of the engineered features over the training set was very similar for the three sentiment classes.

sentiment	punct_ratio	num_consec_punct	capital_ratio	num_words_before_preproc	num_words_after_preproc	avg_word_len_before_preproc	avg_word_len_after_preproc	stop_word_ratio
negative	0.054953	0.529752	0.041340	13.473204	8.208199	4.363381	4.342662	0.285801
neutral	0.053076	0.448732	0.043630	12.342867	7.521587	4.473515	4.343495	0.283306
positive	0.052964	0.477278	0.048681	13.109881	8.115358	4.549553	4.467203	0.274992

Figure 3: Average values for engineered features over training set for each of the three sentiment classes

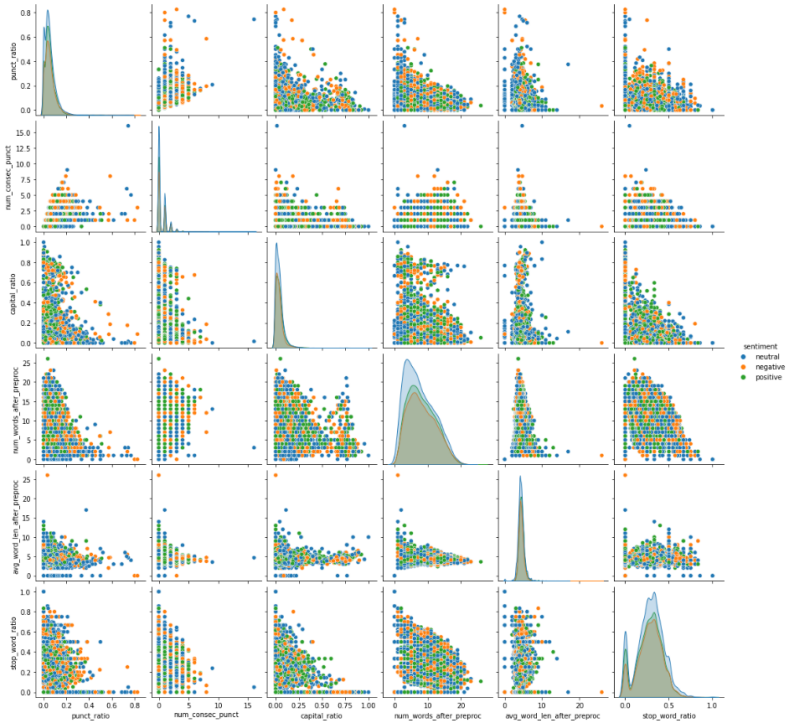


Figure 4: Pairplot for engineered features

4.2 VADER

While researching ways to do sentiment analysis, a non-machine-learning method was shown, which was in the Natural Language Toolkit module of Python - VADER (Valence Aware Dictionary for sEntiment Reasoning).

Vader is a NLP algorithm that checks for particular positive and negative words as well as grammar rules, and scores the text based on such rules (Hutto & Gilbert, 2014; Manoj, 2020). It is not a Machine Learning solution, as it is a rule-based algorithm instead, but it was surprisingly accurate, when compared to earlier machine learning models with only LSTMs.

Preprocessing

VADER does not do processing, it will directly look at words and assign scores to the word, and sum it up, and return the result.

Training and Evaluation

There is no training involved with VADER, as the rules are hard-coded. Evaluation wise, for our dataset it was able to get 66% accuracy, but when we ran it with real-life examples, there were very obvious and interesting mistakes it made, but it was generally on-point when it comes to extremely positive text and extremely negative text.

4.3 TF-IDF

TF-IDF stands for term frequency inverse document frequency. It's a statistic that one can calculate for each word in each document. Multiply the number of times the word appears in the current tweet by the inverse of the number of times it appears in all tweets (Scott, 2021). TF-IDF is a measure of the relevance of a word to the tweet it is in. If a word appears several times in a tweet and few times in all the tweets in the dataset, it is very relevant to that tweet.

Preprocessing

TF-IDF can be calculated for each word in each document. This is done instead of just counting the number of occurrences of a word in the tweet as in the previous method for the baseline models. TF-IDF captures importance in addition to frequency.

Training and Evaluation

The baseline models were again trained, but this time on the transformed TF-IDF text feature vectors of the training set. The multinomial naive bayes linear SVM, and logistic regression achieved 62.4%, 67.7%, and 69.3% average k-fold accuracy respectively. Some models had higher accuracies than with just word counts as features, but no new high accuracy was achieved.

4.4 Word Embedding Neural Network

Unlike a single occurrence count or TF-IDF value, word embedding provides a way to represent a word as a vector of numbers. A word is represented as a vector, or embedding, in N-dimensional space. Words with similar meaning will be closer in space than those with different meaning.

We tried training a word embedding from scratch, but received poor accuracy since our dataset is relatively small. The pre-trained word embedding we used was trained on a much larger dataset and was better able to learn which words are similar.

The pre-trained word embedding we used was created by Stanford's GloVE. GloVE, Global Vectors for Word Representation, is an algorithm for training a word embedding. The pre-trained embedding was trained on a large corpus of two billion tweets.

The word embedding can serve as the first layer of a neural network. In this way, text with words goes into the embedding layer and is turned into embedding vectors. These vectors are then passed to the next layer in the model. In addition to the embedding layer, our model used an LSTM layer. The LSTM layer is actually a network in itself.

LSTM stands for long short term memory. It is a kind of recurrent network, meaning it stores information or context from previous passes of data through the layer for a certain amount of time. LSTM is less susceptible to vanishing and exploding gradients than other recurrent networks. Essentially, it allows for remembering the context and relationship of nearby words. A bidirectional LSTM remembers context from the future as well as the past, by looking at the outputs of future layers as well as past ones. Our network used a bidirectional LSTM layer.

Preprocessing

For this model, preprocessing again broke up a word into tokens. To do this, we used a modified version of a Ruby script used by the developers of GloVe (Pennington et al., 2014). Using their script, we were able to transform our text into tokens the same way they did. This allowed us to generate more tokens that matched the tokens that the pre-trained GloVe embedding was trained on and had learned vector representations for.

A vocabulary was created for all tokens in the training set. After tokenizing a tweet's text, each word was turned into its corresponding index in the vocabulary. These lists of indices were padded to a common length so they could be passed to the model.

Training and Evaluation

To pick the best model, we did hyperparameter tuning using a grid search. We tried different network architectures and different parameters for the layers of the architectures. For each model of the grid search, we did k-fold cross validation on the training set.

The best model had the embedding layer, followed by a dropout layer, a bidirectional LSTM, and a final dense layer for prediction. The average accuracy over the k-fold cross

validation was 76.1%.

	precision	recall	f1-score
neutral	0.720289	0.754952	0.736964
positive	0.817192	0.788619	0.802239
negative	0.763251	0.738052	0.74994
average	0.766911	0.760541	0.763048

accuracy
0.76078

Figure 5: Classification report for word embedding neural network averaged over the folds of cross validation on the training set

4.5 BERT

With traditional LSTM, the frame of memory (context) it can have is maximally 3-5, while with BERT (Bidirectional Encoder Representations from Transformers), it is theoretically able to generate outputs that incorporate the entirety of the words around it. Alongside an embedding, it can generate accurate results as it is now processing each word with considerations of the entire text every time.

Upon research, we have discovered a dedicated database of pre-trained Natural Language Processing models and datasets: HuggingFace, and a framework for us to easily adapt the models to our task. We have decided to use their multi-class classification BERT model and do transfer learning - fit our dataset to it and fine-tune.

Preprocessing

The BERT models we use from HuggingFace are defined in the following way: A tokenizer, just like word embedding, and then instead of using an LSTM layer to process the embedding, a BERT layer is used instead, then a classification layer (Fully Connected Perceptron) is used as a 'head' to generate the class prediction (Yalçın, 2021).

With multi-class classification, we can simply define 'negative', 'neutral' and 'positive' labels as classes, and have the model figure out the category (label) of said queries.

Training and Evaluation

Since this is transfer learning and we are fine-tuning, this model, with its amount of weights and nodes (due to BERT), is very costly to train. Additionally, it is very prone

to overfitting as the huge amount of parameters (BERT, and word embedding) are pre-trained on a huge dataset (unpublished books and the English Wikipedia). Our dataset of 20,000+ entries will greatly upset the weights if trained more than a couple of epochs, even with a low learning rate.

Aside from the normal BERT model from Hugging Face, we have also tried vinai's BerTweet model in an attempt to better fit our dataset and use-case. This model results in faster training and higher accuracy without overfitting (Nguyen, Vu, & Tuan Nguyen, 2020; Delvin, n.d.). Accordingly, our accuracy was able to stabilize at 80% and sometimes reached over 81%.

In the end, we chose to utilize a BERTweet model, fine-tuned for 1 epoch with learning rate 3e-5. Although using BERTweet allows us to fine-tune it for 2 epochs without major overfitting, the computing cost (and the potential bits of overfitting) outweighs the marginal performance gain.

Throughout many trials and errors with the BERTweet model, we have discovered, and concluded, that our dataset has become a limit to how accurate our model can be. The best way to fine-tune BERT (and in extension BERTweet), is to run as many unique entries through it as possible, and do it only once, as during pre-training these models are trained on a dataset magnitudes larger than what we have for the task. This will ensure that the model is updated to both react to dataset keywords correctly in the BERT layer, and to decide correctly in the decision layer ('head'), without over-adhering to our dataset.

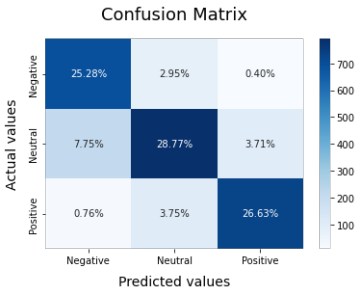
	precision	recall	f1-score
neutral	0.795708	0.719125	0.754437
positive	0.81897	0.866175	0.841036
negative	0.779582	0.832308	0.804488
average	0.798087	0.805869	0.799987

accuracy
0.797226

(a) Classification report for BERT averaged over the folds of cross validation on the training set.

	precision	recall	f1-score	support
0	0.75	0.88	0.81	787
1	0.81	0.72	0.76	1106
2	0.87	0.86	0.86	856
accuracy			0.81	2749
macro avg	0.81	0.82	0.81	2749
weighted avg	0.81	0.81	0.81	2749

(b) Classification report for final BERT model trained on all of training set and evaluated on test set



(c) Confusion matrix for final BERT model trained on all of training set and evaluated on test set

Figure 6: Bert classification reports and confusion matrix

5 Conclusions and Potential Applications

In summary, we have applied various methods of sentiment analysis to our taks, and found that we are able to achieve the best accuracy using pre-trained BERT models. We have given others our serious attempts too, such as GridSearching our second runner: the Word Embedding model. But unfortunately LSTM-based models weren't able to achieve higher than 76% accuracy, which we think is the extent of the structure's capability against our task.

In addition, by manually checking real-life scenario prediction results, we have also discovered that there isn't much subjective difference between the models, that is, the dictionary-based method VADER, and the various simple models can still be considered 'accurate enough' as a general indicator to sentiment. To explain, during manual review of the predictions, what is obviously 'positive' is marked positive, and what is obviously 'negative' is marked negative by all of our models. But, what sets the high-accuracy-models apart is how they define 'neutral', which is what we considered the hardest part of this task. With BERTweet, we did observe that more fringe cases were handled correctly than the other models, although it wasn't able to account for all of the edge cases, as indicated by our confusion matrix.

Though obviously, none of the models here can achieve perfect accuracy, or a fully satisfactory subjective review, as a lot of things, including 'subjective review', and even the labels, can be considered subjective. Hence there is no best answer for everything in our task, and instead we have created something that can be described well enough by its accuracy value: 80%.

But, overall, our final model, perhaps not as state-of-the-art like OpenAI GPT3, produces results that are imperfect, but 'reasonably accurate' as a general sentiment indicator of the replies to a tweet.

For this project, we created a very basic website where a user can paste in any recent tweet, and we will return a graph showing how many positive, negative, or neutral replies that tweet got within the last week. The main goal of our project was designing a good model, but even the basic returned results can be very useful in giving someone a bit more in depth look at a tweet's perception than just likes alone.

However, we believe that this project could be taken vastly further than the demonstration we gave. One small example of this possibility is shown when our API code fetches the replies of a tweet. It not only gives us the reply itself, but also the ID of the user that made it. This little detail is like a door that opens up nearly limitless possibilities for more in depth data. Using these IDs, we can look at all the other public activities the user had on the site, which other profiles their following, and who follows them. We can build our own profiles on each user that replies and tell our client what kind of people had positive/negative replies to their tweet. We can tell which groups those people belong to and predict which communities will react to a certain tweet and how.

For potential customers, there are major, million-dollar-revenue content creators that are spending a sizable amount of their time manually browsing through twitter and manually

reading replies to their published contents, and there are companies that have failed to react in time to a Public Relation emergency, and that made them lose entire departments of operation and millions of dollars of profits. In the modern, internet-connected world, information is money, and time is also money. Twitter, although proven to be problem-ridden by the recent acquisition by Elon Musk, is still the center-stage of the cyber-world, as news would 'break out', and become wide-spread in twitter first, and responses to such are immediate. Although it is feasible to read tweets and replies one-by-one (or pay someone to do it), the time and money consumed in this activity is obviously wasteful, when we have many powerful language analysis tools available today. The project serves as an exploration and development of such a tool to quickly analyze sentiment towards a certain item, in this case, sentiment of replies to a tweet, but many other language analysis methods can be utilized to give a comprehensive view towards any keyword and any situation, within minutes. The goal is that the person using such a product can get an up-to-date general overview of the situation of a keyword / incident / scenario on twitter within minutes, so they can immediately make their decisions before it is too late.

This kind of detailed information would be incredibly valuable, also, to marketing agencies in helping them build ad campaigns or garner some kind of desired publicity. These are the kind of paths we would love to explore more with this project if given the time, but there really is a rabbit hole of possibilities here that you could probably make a career out of.

References

Birjali, M., Kasri, M., & Beni-Hssane, A. (2021, August). A comprehensive survey on sentiment analysis: Approaches, challenges and trends. *Knowledge-Based Systems*, *226*, 107134. Retrieved 2022-06-01, from https://www.sciencedirect.com/science/article/pii/S095070512100397X doi: 10.1016/j.knosys.2021.107134

Catania, L. J. (2021, January). 3 - The science and technologies of artificial intelligence (AI). In L. J. Catania (Ed.), *Foundations of Artificial Intelligence in Healthcare and Bioscience* (pp. 29–72). Academic Press. Retrieved 2022-06-01, from https://www.sciencedirect.com/science/article/pii/B9780128244777000092 doi: 10.1016/B978-0-12-824477-7.00009-2

Delvin, C. (n.d.). *bert-base-uncased · Hugging Face.* Retrieved 2022-04-05, from https://huggingface.co/bert-base-uncased

Gupta, S. (2018, March). *Applications of Sentiment Analysis in Business.* Retrieved 2022-06-01, from https://towardsdatascience.com/applications-of-sentiment-analysis-in-business-b7e660e3de69

Hutto, C., & Gilbert, E. (2014, May). VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. *Proceedings of the International AAAI Conference on Web and Social Media*, *8*(1), 216–225. Retrieved 2022-06-05, from https://ojs.aaai.org/index.php/ICWSM/article/view/14550 (Number: 1)

Manoj, P. (2020, December). *Sentiment Analysis Using Python and NLTK Library.* Retrieved 2022-06-05, from https://medium.com/swlh/sentiment-analysis-using-python-and-nltk-library-d68caba27e1d

Nguyen, D. Q., Vu, T., & Tuan Nguyen, A. (2020). BERTweet: A pre-trained language model for English Tweets. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 9–14). Online: Association for Computational Linguistics. Retrieved 2022-06-05, from https://www.aclweb.org/anthology/2020.emnlp-demos.2 doi: 10.18653/v1/2020.emnlp-demos.2

Pandian, A. (2020). *NLP Beginner - Text Classification using LSTM.* Retrieved 2022-06-01, from https://kaggle.com/arunrk7/nlp-beginner-text-classification-using-lstm

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532–1543). Doha, Qatar: Association for Computational Linguistics. Retrieved 2022-06-05, from http://aclweb.org/anthology/D14-1162 doi: 10.3115/v1/D14-1162

Periwal, N. (2021). *Twitter Sentiment Analysis for Beginners.* Retrieved 2022-05-31, from https://kaggle.com/stoicstatic/twitter-sentiment-analysis-for-beginners

Scott, W. (2021, September). *TF-IDF for Document Ranking from scratch in python on real world dataset.* Retrieved 2022-06-05, from https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089

Taboada, M. (2016). Sentiment Analysis: An Overview from Linguistics. *Annual Review of Linguistics*, *2*(1), 325–347. Retrieved 2022-06-01, from https://doi.org/10.1146/annurev-linguistics-011415-040518 (_eprint: https://doi.org/10.1146/annurev-linguistics-011415-040518) doi: 10.1146/annurev

-linguistics-011415-040518

Yalçın, O. G. (2021, February). *Sentiment Analysis in 10 Minutes with BERT and Hugging Face.* Retrieved 2022-06-01, from `https://towardsdatascience.com/sentiment-analysis-in-10-minutes-with-bert-and-hugging-face-294e8a04b671`