Sentiment Analysis on Twitter Data

Eric Wu, Minjun Park, Angela Yu, Hansen Guo

Professor: Adam Meyers, New York University

Abstract

This paper will explore and compare multiple algorithms for sentiment analysis on posts from Twitter, a micro-blogging platform. Twitter has become an increasingly dominant site where people express their opinions and emotions in a short post, or Tweet, that is limited to 280 characters. Because of its shortened form, it is expected that the positivity of a subject will be more easily obtained. We used two Kaggle datasets that contained Tweets which were manually labeled with people's sentiments to be used as our training dataset and testing dataset.

1. Introduction

Over the past decade, social media and micro-blogging have grown to be prominent tools of communication through which people can express their thoughts. They have become platforms that can provide insight on a wide variety of information, as people broadcast their opinions on the products and services they use, their likes and dislikes, their political views, and much more. As a result, these websites have become data goldmines for companies to gauge how people feel about them and their products or services. A company can improve their product offering by monitoring the general reaction towards its features and developing it to adapt to people's needs and wishes. This has led to a high demand for sentiment analysis by many different entities that want to be knowledgeable about the opinions of the people who affect their success, such as:

- Companies that want to know how people view their products and services.
- Brands that are looking to find influencers to promote their products.
- Political parties that need to know how the general public is responding to their actions and policies.

In this paper we will focus on Twitter, one of the most frequently used micro-blogging platforms today that has over 330 million monthly users. Twitter is the ideal candidate for corporations to mine and analyze data from for many reasons:

- Just like other micro-blogging websites, Twitter is a place for people to share their opinions, and there is no limit to the amount of different topics that are discussed day-to-day.
- With over 500 million Tweets posted everyday, Twitter boasts a tremendous amount of posts to gather information from.
- The amount of people who use Twitter is constantly growing, and the wide variety of its users makes it possible to collect data from people of different social, political, and ethnic groups.

We found a dataset on Kaggle of 27,481 Tweets that were manually labeled as one of three categories (positive, neutral, negative), and used it as training data to compare four different algorithms for classifying Tweets based on sentiment.

In the rest of our paper, we will discuss how we preprocessed our data, how we measured accuracy, and the methods we implemented – Naive Bayes, CountVectorizer, Support Vector Machine (SVM), and Long Short Term Memory (LSTM) – to analyze how successful they were at measuring sentiment.

2. Task Description

Our original goal was to develop a system to handle sentiment analysis in Tweets. However, this seemed like an overdone task, as it has been the topic of focus for many SemEval conferences and real-time twitter sentiment analysis projects. Therefore, after researching more about similar previous works in this field, we shifted the focus of this paper to comparing various different kinds of models to see which best executes Twitter sentiment analysis. The models we chose to specifically have completely different methods of training the data from one another, so it will be easy to compare the success of each one.

3. Datasets

3.1 Preprocessing

Data preprocessing is used to remove any cases of incomplete, noisy, and/or inconsistent data. Especially in cases such as Twitter where anyone can post nearly anything, there is a lot of cleanup necessary before we can feed it to our algorithms to train the data. We started by removing all cases of stop words. These are common words that add no meaning towards the sentiment of the text and often do not convey meaning. We imported the stop words from the Natural Language Toolkit (NLTK) package as the basis for our stop words. Note that the term *not* was included in our list of stop words for Naive Bayes, CountVectorizer, and SVM because they work badly with negation, but reintroduced in our deep learning model, explained in 5.3. In a similar fashion, cases of $n't$ were removed originally, but reintroduced later in our deep learning model. A more extensive list of our preprocessing steps are listed below:

- Removal of stop words: The stop words were obtained through NLTK's list of stop words. These words were removed completely when preprocessing the data.
- Dealing with negations: Negations are expressions such as "didn't" that may change the sentiment of a sentence. For example, sentences such as "I didn't like it." gives a negative sentiment, but may be classified as positive. Two possibilities were explored. The first possibility was to remove all words that contained "n't" which meant that it needed to rely on other words within the sentence for proper sentiment analysis. The second option that was explore was to add "NOT_" before all words after negations until a punctuation is reached.
- Removal of user handles starting with @: All words containing @ were replaced with the text "AT_USER" to prevent username handles from affecting the classification of a tweet.
- Removing numbers and special characters: A basic regex algorithm was used to remove

numbers and special characters from the tweets

- Conversion of emoticons: Emoticons can give valuable information about a tweet. A smiling emoticon may imply that the tweet has a positive sentiment yet a frowning emoticon can give the tweet a negative sentiment. Thus, all emoticons were converted to text "emote_positive" and "emote_negative". Emotes that were not relevant (such as a hamburger emoticon) were removed.

- Removal of URLs: A basic regex algorithm was used to remove all URL in tweets.

- Lemmatize certain words: Lemmatizing words is important to ensure that words that are same but have different forms (i.e. studies and studying) are changed to be the same word (study). This was done through NLTK's lemmatizing function.

- Stem certain words: Similar to lemmatizing, stemming words allow us to consolidate different words that may mean the same thing into one. Stemming may be done in place of lemmatization through nltk.stem's PorterStemmer class.

- Fix spelling errors: SymSpell, which is based on the Symmetric Delete spelling correction algorithm, was considered for usage. However, due to the vast amount of words, using a function to check each spelling consumed a lot of time, thus it was removed from some of the algorithms tested.

- Converting text to lowercase: This was done through python's function l̇ower().

Note that that lemmatizing and stemming were imported from nltk.stem's WordNetLemmatizer and nltk.stem's PorterStemmer respectively.
Note that fixing of spelling errors was imported from Pattern's spelling library.

3.2 Training Data

The training data consisted of 27,481 tweets in a Comma Separated Value (CSV) file obtained from Kaggle. There were two columns: the first column consisted of the raw Tweets and the second column consisted of the sentiment label. If we were to remove the 11,119 neutral-sentiments, we'd have a testing dataset size of 16,362 tweets.

1	I`d have responded, if I were going	neutral
2	Sooo SAD I will miss you here in San Diego!!!	negative
3	my boss is bullying me...	negative
4	what interview! leave me alone	negative

Fig. 1.— Example of four Tweets in the training data set

3.3 Testing Data

The test data consisted of 2,534 Tweets in a CSV file obtained from Kaggle. There were two columns: the first column consisted of the raw Tweets and the second column included their corresponding sentiment labels. If we were to remove the 1,430 neutral-sentiments, we'd have a testing dataset size of 1,105 tweets.

3.4 Development Data

The development data used consisted of 300 tweets in a CSV file obtained from Kaggle. There were two columns: the first column consisted of the raw Tweets and the second column included the sentiment label.

4. Scoring

Because both our training data and our testing data already had labelled sentiments, we did not have to face the issue of having to crowdsource the sentiments of our tweets. Therefore, after training the algorithm on the training data and then applying the model on top of the training model, we could easily just go down the list and check whether or not the model accurately predicted the sentiment for a tweet. The accuracy of our model can be measured simply from:

$$\text{Testing Accuracy} = \frac{\text{Number of tweets accurately predicted}}{\text{Number of tweets}} \tag{1}$$

5. Models

5.1 Naive Bayes

Naive Bayes Classifer relies on Bayes' Rule, which gives the probability of an event occurring based on a previous event. Given events A and B, the probability that event A would happen based on event B can be written as:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2}$$

where $P(B|A)$ is the probability that event B would occur if event A were true and $P(A)$ and $P(B)$ are the independent probabilities of A and B. When estimating the right sentiment $\hat{c}$ in document d, we are able to form an equation:

$$\hat{c} = \underset{c \in C}{argmax} P(c|d) \tag{3}$$

Substituting the equation with Bayes' rule then gives us

$$\hat{c} = \underset{c \in C}{argmax} \frac{P(d|c)P(c)}{P(d)} \tag{4}$$

However, as the set of documents will be constant, we are able to remove the denominator to give us a simplified equation

$$\hat{c} = \underset{c \in C}{argmax} P(d|c)P(c) = \underset{c \in C}{argmax} P(f_1, f_2, \ldots, f_n|c)P(c) \tag{5}$$

The Naive Bayes Algorithm is then based on two assumptions to make this calculation easier. One assumption is that the position of the features do not affect the classification. The second assumption allows for each probability $P(f_a|c)$ to be computed independently from each other.

Relating this to sentiment analysis, there are several steps that needs to be taken. Data must first be processed in order to not only make our classification more accurate but also to save memory

by removing texts such as stop words. These preprocessed texts were then stored in a list of tuples. Each tuple contains a list of relevant words within the text as its first item and a sentiment label as the second item. This organization helps later on when generating bag of words and applying features into these words.

Then, a bag of words must be created – satisfying our first assumption – where each words in our training set will be in. This step was fairly simple as all the words in the training set were appended in a list. The frequency of the words within a list was then calculated using NLTK's FreqDist command, which returns a list of words with keys as their frequency within the bag of words. Creating this bag of words allowed us to match our existing vocabulary with tweets to be analyzed, ensuring all the available features are applied.

We then generate a feature vector that the Naive Bayes Classifier can use. The feature vector will be generated using NLTK's applyfeatures command which returns an object that acts like a list but saves memory by not storing all the feature sets in memory, which is useful when dealing with large amount of data. The training portion of the algorithm was then acheived using NLTK's NaiveBayesClassifier class' train function. We then classified our testing set through its classify function.

5.2 CountVectorizer

Another method that we evaluated is a vector method utilizing the CountVectorizer feature in scikit learn (Pedregosa et al., 2011). The method creates a unigram model by keeping track of the number of occurrences of each word and then weighing the words based on how often they show up in certain kinds of categories. Once a unigram model is created, the algorithm would then search all subsets of the test Tweet and calculate a score based on the tracked weights. The sentiment would then be given based on the weights.

To find the weights, we would first for each subcategory $j \in \{positive, neutral, negative\}$, find all the words i in the tweets belonging to class j. After that, calculate $n_{i,j}$ = the number of tweets in class j containing word i. Let d_j be the number of tweets in class j. The proportion of the tweets in class j that contain the word i would therefore be

$$p_{i,j} = \frac{n_{i,j}}{d_j} \tag{6}$$

Then, we can assign the weights to each word within the class.

$$w_{i,j} = p_{i,j} - \sum_{k \neq j} p_{i,k} \tag{7}$$

With the weights, we can use it to calculate the sentiment based on the subsets of words in the tweets. Let j be the sentiment of the tweet, for each subset of words in the tweet, calculate $\sum_i w_{i,j}$, where i is the set of words in the tweet. After, return the subset of words with the largest sum, given that it exceeds some threshold. With such, the algorithm managed to achieve around a 66.5% accuracy on the development set, and 65.1% accuracy on the test set.

5.3 Support Vector Machine

Another algorithm that we looked into was the Support Vector Machine (SVM) model, which is a machine learning model for two-group classification problems. This meant that our initial goal of classifying based on positive, neutral, and negative sentiments needed to be changed for this algorithm. We did this by removing all cases of neutral sentiments in both the training dataset and the testing dataset to ensure that there were no neutral statements to choose from. From there, we extracted features from the Tweets that would guide the machine learning process. See figure 2 for an overview of the process.

We used a term weighting scheme in order to extract the most classical features as an input to the classifier. Following a similar path as that of University Teknologi Malaysia's paper in the 2014 IEEE International Conference on Computer, Communication, and Control Technology, we applied a term-weighting schemes, namely Term Frequency Inverse Document Frequency (TFIDF) (Zainuddin et al., 2014). TFIDF, which normalizes the resulting vector of each document, can be calculated as follows:

$$\text{TFIDF} = \text{Term Frequency} * \log\left(\frac{\text{Number of tweets}}{\text{Number of tweets containing a certain term}}\right) \tag{8}$$

We then used a feature selection method for our SVM model, specifically Chi-Square. While this test is typically used in statistics, it can also be used in the context of feature selection to examine how often a term or class appears, and is used to measure the deviation of the expected counts versus the observed counts. CHI2 can be computed with the following equation:

$$CHI2(t, C) = \sum_{t\epsilon\{0,1\}} \sum_{C\epsilon\{0,1\}} \frac{(N_t, C - E_t, C)^2}{E_t, C} \tag{9}$$

In this formula, N and E respectively refer to the observed and expected frequency for term t and class C.

After implementing the CHI2 feature selection with the TFIDF feature extraction, the accuracy improved greatly and our final accuracy score with the SVM model was 73%.

5.4 Deep Learning Model

The algorithms that we incorporated earlier were those that of machine learning. However in the previous cases that use Bag of Words, using single one-hot vectors, we already see conflicts with Twitter's dataset. For example, due to the innate data sparsity in Tweets which often have irregular, misspelled, and/or abbreviated words, as well as with the extended difficulty with the character limitation, it is difficult for simple machine learning to correctly identify positivity. Furthermore, simple machine learning loses sequence information which results in failure to identify polarity properly (Pang et al., 2002). In cases such as when a negation phrase is used, algorithms using Bag of Words would express an opposite sentiment to that of the original statement. This is because it falsely identifies words with special functions in tasks that are not distinguished in these algorithms. This is problematic for sentiment classifiers as it disregards phrases, instead of words. In order to capture the meaning of an entire sentence, it is necessary to analyze complex sentiment

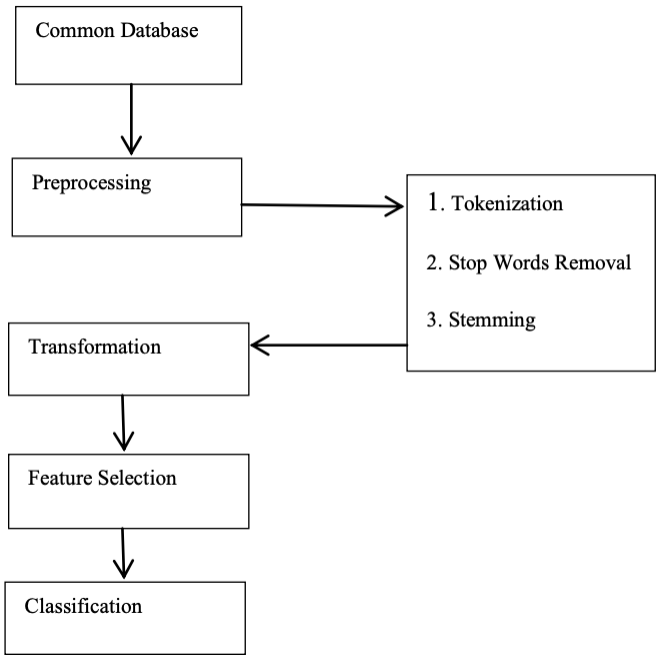

Fig. 2.— Depicts the process that our SVM model goes through. Note that TFIDF occurs during transformation

expressions. This is why in our original proposal we had suggested to analyze a deep learning algorithm similar to that of Harbin Institute of Technology's sentiment analysis program used in SemEval-2013 convention(Wang et al., 2015).

We figured that because this was an extension of our project, it would be more time efficient to start with an open source project (Bhanot, Karan, 2019). This project contains a different dataset, and uses a Long Short Term Memory (LSTM) recurrent neural network (RNN) to train the data, allowing for the ability to consider how words interact with one another in the training process. Note that a simple RNN model would have failed to capture the context information of a set of complex interactions among words. We use a LSTM model specifically because it allows for memory blocks, which contains self-connected memory cells, input, output and forget gates, as seen in Fig 3, to solve the aforementioned issue of simple RNN models.

This should allow the algorithm to process entire sequences of data – in our case a whole Tweet – instead of only being able to process single words. By doing so, in cases such as negation phrases, the model should not produce a result opposite to that of the original sentiment.

Open Source Model

There were definitely some drawbacks to starting off with an open source project, one being that we had less customization over the data. The original code had contained the code to process their data, engineer features, and perform sentiment analysis. While it would be nice to directly pull from this project, we wanted to apply our own data as well as execute our own preprocessing on the dataset. Furthermore, their data had only classification of positive and negative, while it was our original goal to have three classifications of negative, positive, and neutral. This led us with

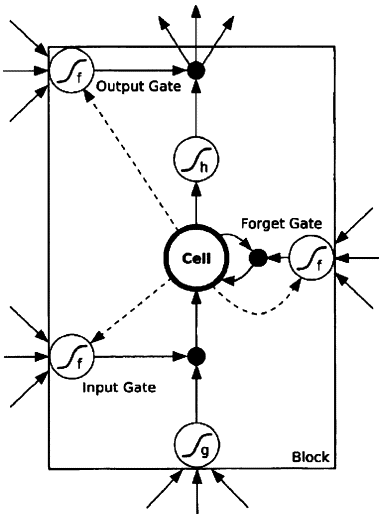

Fig. 3.— Image of what a LSTM memory block looks like with with a single self-connected memory cell. The Input and Output Gates measure and add/remove a scaled version of the input and output respectively to the internal activation, or state. The Forget Gate can reset the state (De Muler, Wim, 2014)

the conclusion that there were three main variables that differed in this project to our original goal:

- Using the correct dataset/preprocessing
- Splitting testing/training data
- Positive/Negative vs Positive/Negative/Neutral

The main thing we wanted to pull from their project was the way they use the Deep Learning Keras package (an open-source neural-network library written in Python) to train their data. However, before building our own model, we wanted to test how their project would run with their own data and preprocessing. The result was a very high testing accuracy of 79%. While this value is quite high in comparison to our original models of Naive Bayes or CountVectorizer, we must understand that they had different ways of preprocessing and were only classifying based on two results (positive and negative).

Our Model

First and foremost, we wanted to switch datasets and apply our own preprocessing to the data. The open source model also used a dataset from Kaggle, but it contained 1.6 million tweets. We wanted to do a similar procedure, but instead apply the algorithm they used to our dataset. As noted in section 3.1 Preprocessing, there were some features that we had wanted to keep in the preprocessing for this model. For example, cases of *not* were removed from the stop words to be kept in the text. Furthermore, contractions with $n't$ were expanded to be *not*. This way negation could be detected by the algorithm, rather than ignored in cases of Naive Bayes and CountVectorizer.

The next step was splitting our testing and training data. While the open source model had randomly split its 1.6 million tweets into 80% training and 20% testing data, we wanted to maintain

our original testing and training data. Thus the new testing and training data remained the same as our previous algorithms.

The final issue was that of neutral sentiments. We had originally wanted to create an algorithm that would work with neutral sentiments, however, programming our own LSTM model was a lot more challenging than originally intended. Therefore, we instead removed all cases of neutral sentiments in both the training dataset as well as the testing dataset, similar to how we did with the SVM model. This way, the LSTM model would work with our dataset.

The LSTM model that was used was from Keras, a Python deep learning API, and ultimately gave us a testing accuracy of 73%.

6. Results and Analysis

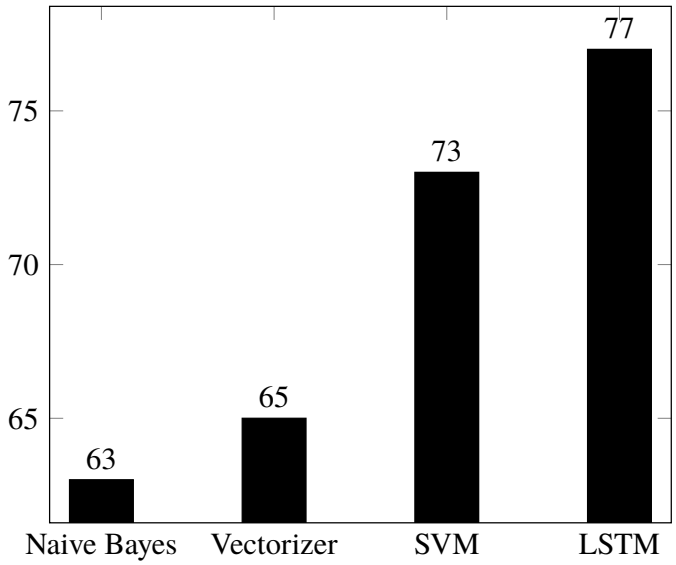

Fig. 4.— Graph of the testing accuracy of all four algorithms

As shown by Figure 4, we see that our SVM and LSTM models have larger testing accuracies than both Naive Bayes and CountVectorizer, this is almost definitely because the former are two-group classifications, which means we're only classifying based on positive and negative sentiments. Meanwhile, in the latter cases, we were classifying based on three sentiments of neutral, positive, and negative; this makes it more likely to run into errors. A comparison between these different classifications would be unfair, and therefore in our analysis, we'll specifically be comparing Naive Bayes with CountVectorizer and SVM with LSTM.

The Naive Bayes algorithm yielded the lowest percentage out of all the different models. We believe that one of the reasons for this lies behind the assumptions that Naive Bayes states that the position of features do not matter. Sentences such as "I do nothing to enjoy life" and "I enjoy nothing in life" give very different sentiment with similar words in different orders. With grammar being crucial in every day language, making the blind assumption about the positions of features not mattering may have affected the algorithm. These assumptions that were meant to help calculate efficiently may have affected the accuracy of the results.

The CountVectorizer model is similar to the Naive Bayes model, in that both end up using a bag of words. Likewise, the CountVectorizer algorithm does not account for the order of words in its algorithm, and is therefore equally vulnerable to some of the same issues that caused the Naive Bayes algorithm to yield the relatively low accuracy. Where CountVectorizer overcomes Naive Bayes is that it uses its ad-hoc algorithm for classification. This classification difference can be alluded to in the two percent increase in accuracy between Naive Bayes and CountVectorizer.

In the SVM model, the implementation of TFIDF with the Chi-Square feature selection method produced a testing accuracy of 73 %, which when considering that it has a 50% chance of guessing the sentiment correctly at random, isn't great. However this can be greatly attributed to the lack of testing data where there wasn't enough data for the algorithm to have a properly trained classification.

Similar to the SVM model, the LSTM algorithm faces a similar crux of the lack of training data. However, where we can attribute the success of the LSTM algorithm over its counterpart is through its ability to provide more flexibility and simulate interactions between word vectors. This algorithm is able to handle cases of negation which would have been previously ignored by the other three models, as negations would have been removed in the preprocessing step. Recurring Neural Networks, specifically LSTMs, tend to have the most success when working with sequences of words and thus are a perfect fit for Twitter data. If we were to have increased sizes of datasets, its likely we would see a better and more well-trained model which could end up increasing the testing accuracy greatly.

If all four algorithms were to be compared to one another in our dataset, with neutral cases all removed, we could expect CountVectorizer and Naive Bayes to have a better score than that of SVM and LSTM; though, with a large enough dataset, we believe that SVM and LSTM would be able to outperform them eventually.

7. Project Breakdown

We each worked on a different algorithm.

- Naive Bayes - Minjun Park
- CountVectorizer - Hansen Guo
- Support Vector Machine - Angela Yu
- Long Short Term Memory - Eric Wu

8. Extensions

Some extensions that we can definitely look into is comparing all four algorithms together. We ran into some issues trying to get CountVectorizer to run two-type classifications, so we decided that it would be best to compare Naive Bayes with CountVectorizer and SVM with LTSM. However, for a future exploration, a good starting point would be to compare the testing accuracy of all the

algorithms. As noted in our analysis, we can expect Naive Bayes and CountVectorizer to be more advantageous than our other models with our dataset; however, given a larger dataset, it is to be expected that the LSTM and SVM would be more advantageous than their counterparts.

This leads us to our next extension, which could be to increase our training data size. Because we were limited in the size of the training data provided by Kaggle, it's likely that SVM and LSTM models were unable to do their more extensive training of data with our limited datset. However, what we can expect is that the larger the dataset, the greater performance of the two algorithms. Something we can do in the future is to look into Sentiment140's Twitter datset with over 1.6 million tweets and sentiments and use that as our dataset instead.

References

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan.2002.Thumbs up?:sentimentclassification using machine learning techniques. In Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10, pages 79–86. Association for Computational Linguistics.

Dan Li and Jiang Qian, "Text sentiment analysis based on long short-term memory," 2016 First IEEE International Conference on Computer Communication and the Internet (ICCCI), Wuhan, 2016, pp. 471-475, doi: 10.1109/CCI.2016.7778967.

De Mulder, Wim, (2014). A Survey on the Application of Recurrent Neural Networks to Statistical Language Modeling. Computer Speech and Language. 30. 10.1016/j.csl.2014.09.005.

Değer Ayata, Murat Saraçlar, Arzucan Özgür, "Political opinion/sentiment prediction via long short term memory recurrent neural networks on Twitter," 2017 25th Signal Processing and Communications Applications Conference (SIU), Antalya, 2017, pp. 1-4, doi: 10.1109/SIU.2017.7960733.

Jurafsky, Daniel and Martin, James (2019) Speech and Language Processing, https://web.stanford.edu/ jurafsky/slp3/4.pdf

Karan, Bhanot, Twitter Sentiment Analysis, (2019), Github repository, https://github.com/kb22/Twitter-Sentiment-Analysis

M. S. Neethu and R. Rajasree, "Sentiment analysis in twitter using machine learning techniques," 2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT), Tiruchengode, 2013, pp. 1-5, doi: 10.1109/ICCCNT.2013.6726818.

Meylan Wongkar and Apriandy Angdresey, "Sentiment Analysis Using Naive Bayes Algorithm Of The Data Crawler: Twitter," 2019 Fourth International Conference on Informatics and Computing (ICIC), Semarang, Indonesia, 2019, pp. 1-5, doi: 10.1109/ICIC47613.2019.8985884.

Niharika Kumar, "Sentiment Analysis of Twitter Messages: Demonetization a Use Case", Computational Systems and Information Technology for Sustainable Solution (CSITSS) 2017 2nd International Conference on, pp. 1-5, 2017.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, D., Brucher, M., Perrot, M., Duchesnay, E. (2011). Scikit-learn: Machine Learning in PythonJournal of Machine Learning Research, 12, 2825–2830.

Priyanshu Jadon, Deepshikha Bhatia, Durgesh Kumar Mishra, "A BigData approach for sentiment analysis of twitter data using Naive Bayes and SVM Algorithm," 2019 Sixteenth International Conference on Wireless and Optical Communication Networks (WOCN), Bhopal, India, 2019, pp. 1-6, doi: 10.1109/WOCN45266.2019.8995109.

Sahar A. El Rahman, Feddah Alhumaidi AlOtaibi, and Wejdan Abdullah AlShehri, "Sentiment Analysis of Twitter Data," 2019 International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 2019, pp. 1-4, doi: 10.1109/ICCISci.2019.8716464.

Xin Wang, Yuanchao Liu, Chengjie Sun, Baoxun Wang, and Xiaolong Wang.2015.Predicting Polarities of Tweets by Composing Word Embeddings with Long Short-Term Memory. In Proceedings of the 53rd

Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 1343–1353, Association for

Y. Woldemariam, "Sentiment analysis in a cross-media analysis framework," 2016 IEEE International Conference on Big Data Analysis (ICBDA), Hangzhou, 2016, pp. 1-5, doi: 10.1109/ICBDA.2016.7509790.

Zainuddin, Nurulhuda Selamat, Ali. (2014). Sentiment analysis using Support Vector Machine. I4CT 2014 - 1st International Conference on Computer, Communications, and Control Technology, Proceedings. 333-337. 10.1109/I4CT.2014.6914200.