Sentiment Analysis for Text Messaging

Abstract:
According to CTIA's press release in July 2019 Americans sent 63,000 texts per second[1] which mounts to one hundred ninety-eighty billion seven hundred two million texts in 2019 alone. In this project, we attempted to classify text messages based on the sentiment conveyed through the text. This process better known as sentiment analysis or opinion mining has gained much popularity through recent advances in technology. In this paper we aim to provide thorough performance analysis of two popular machine learning algorithms namely Support Vector Machines (SVM) and Logistic Regression, while conducting Multinomial Naive Bayes classifications for baseline solution. These models are then trained and tested from a dataset of 1.6 million tweets[2]. Using various alterations of the developed models, we found that Logistic Regression and SVM both outperform the baseline Multinomial Naive Bayes classifier. SVM with Tf-idf (ngram = 1 - 3, min df = 2) performed the best with accuracy 81.08%, with Logistic Regression with count vectorizer (ngram = 1 - 2) coming second at accuracy 80.93%. These models can be used to support development of prototypical social media sentiment analysis platforms.

Introduction:
The project is to apply Natural Language Processing techniques for performing sentiment analysis in a prototypical messaging application. After sufficient training on user texts, the goal is to have a Machine Learning classifier that is able to accurately identify messages as positive or negative. This would require us to detect various emotions from the text, and chat bubbles would modify their color accordingly. For example, positive messages can be shown in blue and negative in red.

There has been significant use of sentiment analysis on various social media platforms. It is frequently used by companies to assess their reputation through various outlets such as user comments, product reviews, and blog posts. However, most of the research and corresponding investigation has been done on longer text documents. Important data regarding public opinion can be found on social media platforms that by nature predominantly involve shorter text. A program like this will allow users to quickly grasp the thoughts and sentiments of the sender. From such data we can even formulate timelines of mood changes, which can be useful in understanding how certain social relationships of the user have evolved over time. A case where

[1] "CTIA Updates Messaging Principles and Best Practices to Further Protect Messaging from Spam." CTIA, July 19, 2019. https://www.ctia.org/news/ctia-updates-messaging-principles-and-best-practices-to-further-protect-messaging-from-spam.
[2] KazAnova. "Sentiment140 Dataset with 1.6 Million Tweets." Kaggle, September 13, 2017. https://www.kaggle.com/kazanova/sentiment140.

such a tool could be applicable is the ability of users to prove their partner has been more negative in their interactions towards them over text.

Survey of Related Work:

Sentiment analysis has slowly entered the social media market through enhanced visualization of text messages in particular. Facebook's Messenger App, for instance, identifies keywords based on which the receiver is presented with a graphic visual, as in the case of 'congratulations' the recipient's screen shows a burst of confetti. Furthermore, it is known that human reactions alter with changes in color. Most messaging apps only allow users to choose a particular theme for their chat GUI. We think that because of this unchanging environment recipients are not able to truly engage in the sender's conversation. By allowing text bubbles to hold meaning full colors we would allow users to dive into a more enhanced form of communication.

Most existing research uses lexicon based approaches to extract the sentiment score of messages, which has been shown to be robust. Researchers have seen similarity between SMS messages and twitter tweets due to the size limitations which results in frequent abbreviations.[3] Hence, twitter data can also be used for training. Noisy text continues to be a challenge to the analysis, since there are frequent spelling and grammatical mistakes or slang present in informal communication compared to text that is extracted from published sources.

Go, Bhayani and Huang (2009) is popular for its use of distant learning to acquire sentiment data. [4] Due to the difficulty of hand labelling a large amount of tweets, they use distant supervision, where the training data contains tweets with emoticons. For example, tweets containing ':)' or ':-)' will be labelled as positive and those containing ':(' can be labelled negative. The emoticon data is then stripped off for training purposes. Post processing was particularly detailed in this paper, where various techniques were tested for feature reduction. For example removing usernames, urls and repeated letters in words reduced their number of features to 45% of the original. They then built on classifiers using Naive Bayes and Support Vector Machines. The unigram model outperformed everything else, and in their experience, bigrams and Parts of Speech features did not particularly help.

A paper by researchers at Columbia tried different approaches to beat the baseline unigram model, which has proven to be a hard model to outperform.[5] Regarding the data, they preferred used hand annotated data over the previously used datasets. Although this is more work, it

[3] https://hal.inria.fr/hal-01393775/document.

[4] Alec Go, Richa Bhayani, and Lei Huang. "Twitter sentiment classification using distant supervision." Technical report, Stanford. 2009

[5] Agarwal, Xie, Vovsha, Rambow and Passonneau, "Sentiment Analysis of Twitter Data", Association for Computational Linguistics, 2011 https://www.aclweb.org/anthology/W11-0705.pdf

represents a better sample. They also added a dictionary of emoticons and commonly used abbreviations. The addition of these tools and feature analysis in tree based models increased the accuracy of the classifications by 4% over the baselines shown in Go (2010). Note, the use of twitter specific features and emoticons can be attributed to most of the increase in accuracy, which was ignored in other works that focused solely on linguistics.

Formulation:

TWEET REPRESENTATION:

The task of the project is a binary classification of Tweets into two output classes: positive and negative. In order for the proposed Machine Learning algorithm to be trained and tested on the dataset, each Tweet has to be represented mathematically. Let *t* be a Tweet in the corpus of Tweets T. Each word w_i (for i up to length of *t*) is tokenized and represented as a vector v_i. Each t is accompanied by a label of being 'positive' or 'negative.' The classification model will be trained using the vector representation and corresponding label of each tweet and tested on a sample dataset.

$t_i \text{ in } T \Rightarrow w_1 + w_2 + w_3 \ldots w_{i-1}$ *vectorized to* $t_i \text{ in } T \Rightarrow v_1 + v_2 + v_3 \ldots v_{i-1}$

MACHINE LEARNING METHODS:

1) Logistic Model:

In the case of a binary classification problem, the logistic model will be the same as a maximum entropy model. It does not make any independence assumptions like Naive Bayes, which means with bigrams and trigrams there is no issue of features overlapping. The model is represented as follows:

$$P(Y_i = 1) = \frac{e^{\sum B_i x_i}}{1 + e^{\sum B_i x_i}}$$

Where Y is the category, with 1 for positive and 0 for negative. B_i is the weight assigned to the feature and x_i is the value of that feature in the given tweet. For the example of our sentiment analysis, a feature could be the word 'love'. Then, our x_i would be whether or not that word appears in the given tweet and B_i will be a weight assigned to it. In this case the weight should be positive because containing the word 'love' will increase the probability a tweet is positive. To assign these weights, the model uses a maximum likelihood estimation. There are various solvers for this, in our implementation we restrict our choice to Sklearn's default LBFGS (or Limited memory BFGS) solver.

2) SVM:

For classification, Support Vector Machines find a hyperplane (represented by *w, b*) such that the margin σ between the input data point is maximized.

$$\sigma = y_1(\overline{\omega} \,.\, \overline{x_1} + b)$$

However, the size of the margin can be arbitrarily large if a large enough value of ω is selected. ω is constrained in the following optimization problem:

$$min_{\overline{\omega}} \tfrac{1}{2}\overline{\omega}.\overline{\omega} \qquad s.t. \quad y_1(\overline{\omega} \,.\, \overline{x_1} + b) \geq 1 \cdots y_n(\overline{\omega} \,.\, \overline{x_n} + b) \geq 1$$

A visual representation of how SVM's work is shown below:

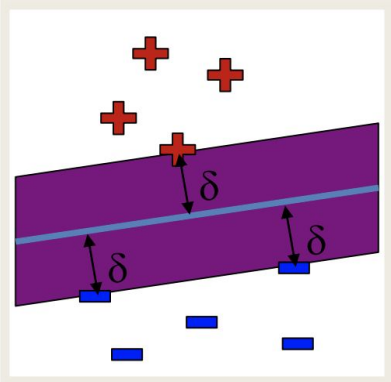

Figure 1: A hyperplane formed between input data to be classified. The hyperplane (blue line) seeks to maximise the size of σ which is the margin between the input data points represented by '+' and '-'. [6]

SVMs perform well when a clear distinction between input data to be classified does not exist (thus necessitating the need to find a hyperplane with max σ) and linear classification algorithms fail to provide a meaningful result.

3) Multinomial Naive Bayes:

Considered as a baseline solution in most scenarios the Multinomial Naive Bayes algorithm uses conditional probability calculations to predict a probability distribution for a given set of classes. As is suggested by the name of this classifier, it makes use of Bayes Theorem, a formula relating the conditional probabilities and the actual probabilities:

$$P(A|B) = \frac{P(B|A) \,.\, P(A)}{P(B)}$$

This is used to derive the Multinomial Naive Bayes model which is represented as:

[6] Joonsuk, Park. Lecture Slide "21_MachineLearning_4.pdf". Accessed May 1, 2020. https://piazza.com/class_profile/get_resource/k5chs59eyxk356/k93alg6d7du4ls.

$$P(x|C_k) = \frac{(\sum_i x_i)!}{\Pi_i x_i!} \Pi_i p_{ki}{}^{x_i}$$

We see that the probability estimation is directly proportional to the number of occurrences of a feature in the data set. Hence, if a feature has never occured in a particular class in the training set and is encountered in the test set, the probability of that goes to zero which causes every other feature in the class to become to have a zero probability.[7]

Approaches:

TEXT PROCESSING:

By tokenizing, normalizing and noise removal we are able to improve the performance of our models as text preprocessing brings in consistency, and removes unnecessary information from the text. Our training data is firstly normalized to lower cases only, before being tokenized to form a list of words.

The next step included the removal of stop words such as usernames starting with '@' sign, email addresses, websites which areThis project uses python3.7 which allowed us to use libraries which include csv, nltk, numpy, os, pandas, pickle, re, scikit-learn, tkinter. The csv and os library enabled us to import datasets while numpy and pandas library allowed us to work with data structures such as Data Frame for the datasets and further the pickle library allowed the saving of pickled files. The nltk library provided us with useful functions to preprocess text and scikit-learn library provided us with the machine learning algorithms. Finally tkinter library aided in the development of the GUI.

Using a dataset of 1.6 million tweets we had 800,000 positive and 800,000 negative tweets at our disposal. Due to limited computing power available on our personal computers we restricted our choice to 25% of the dataset. By following an 80-20 approach to validation, we have 320000 tweets to train and 80000 tweets to test our classifiers. This random subset of the sentiment140 dataset originally introduced in Go (2010) seems sufficient for the learning task.
a common part of text messaging conveying no information relating to the sender's sentiment.

We tested removing stop words using the stopwords() method present in the nltk library. This method removes a larger number of stop words based on its default list which includes negations such as 'nor, 'not' and even 'no'. Given the short-text nature of tweets, valuable information is lost if stopwords are removed using the nltk library. For example, "I am not happy" becomes

[7] "Naive Bayes Classifier." Wikipedia. Wikimedia Foundation, Accessed May 1, 2020. https://en.wikipedia.org/wiki/Naive_Bayes_classifier.

"happy," after removal of stopwords. With regard to semantics, this can adversely affect the performance of the model.

Further to increase the performance of our models in scenarios where it encounters a new word, we stem the training data set so that it only contains the root form of the words. Having two stemmers available in the nltk library namely the Porter Stemmer and Snowball Stemmer. We selected the Snowball Stemmer based on the claim made on the stemmer's documentation that the Snowball('english') is better than the Porter Stemmer.[8]

Therefore it was concluded that the best procedure for noise removal on our data would be to remove customized stop words, and stem them using the Snowball Stemmer.

Experiments & Analysis:

MULTINOMIAL NAIVE BAYES:

A baseline MultinomialNB is used as a benchmark classifier for comparison to Logistic Regression and Support Vector Machine. The results are shown in Table 1 with the use of both tf-idf and countvectorizer.

LOGISTIC REGRESSION VERSUS SUPPORT VECTOR MACHINE:

In the logistic regression, the only hyperparameter experimented on that yielded significant changes in the performance was the value of 'c', which is the inverse of the regularization strength. The results for the unigram count vectorizer model using unigrams is shown below:

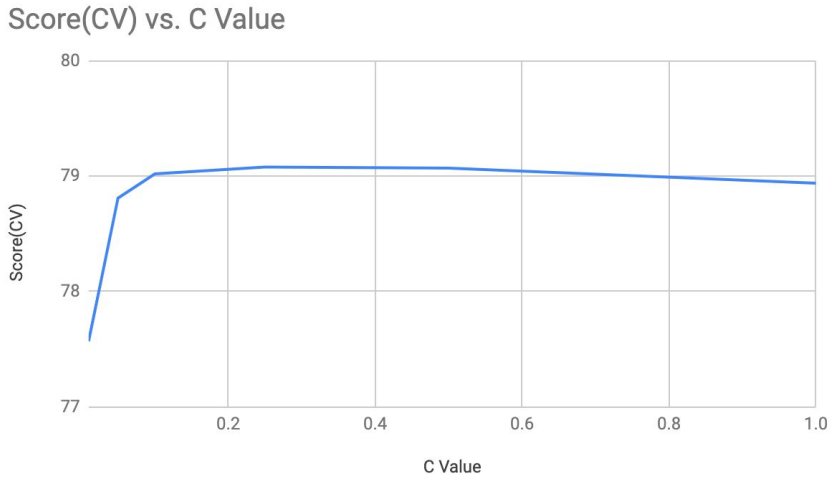

Figure 2: Plot of logistic regression's hyperparameter against performance

From Figure 2 we can see that a value of around 0.25 is optimum for this hyperparameter. The effect of this can be as much as 1.5% jump in accuracy when compared to low values of c, which

[8] "Stemmers." Accessed May 3, 2020. http://www.nltk.org/howto/stem.html.

indicates very strong regularization is counterproductive for this model. We will continue to use this value of c for all logistic model variations.

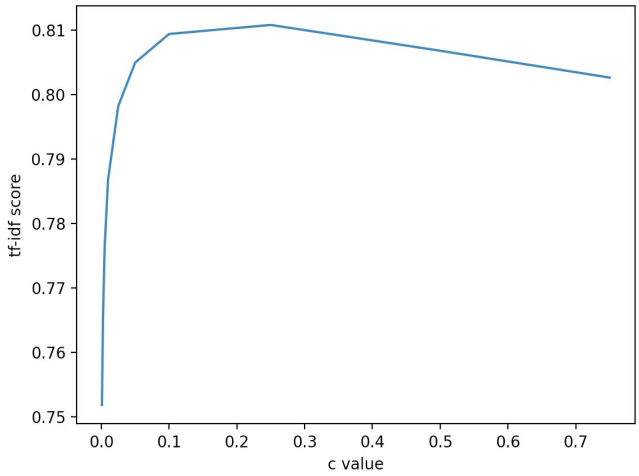

Figure 3: Plot of SVM's hyperparameter against performance

For SVM, a similar investigation was conducted where c values were set in the range 0.001→ 0.5. From the graph plotted above, we can see that a value between 0.2-0.3 is optimum for the inverse of the regularization parameter. Experimentation suggested that setting c = 0.25 yielded better results, and like Logistic Regression, very strong regularization meant results going down. Thus, this value was used for all svm model variations. It is to be noted that Sklearn's LinearSVC was used (SVM with linear kernel) for our experiments, since vanilla SVM with big datasets yield meaningless results.

Table 1 below shows the accuracy scores resulting from several combinations of vectorizors and features for variations of our models.

Model	**Vectorizor**	**Features**	**Accuracy Score**	**Number of Features**
Multinomial Naive Bayes	Count(binary = True)	Unigram	76.99%	92671
	Count(binary = False)	Unigram	76.83%	92671
	Count(binary = True)	Ngrams = 1 - 2	78.69%	1044111
	Tfidf	Unigram	76.12%	92671
		Ngrams = 1 - 2	78.84%	1044111
		Ngrams = 1 - 2, minimum df = 2	78.99%	276648

		Ngrams = 1 - 3, minimum df = 2	79.24%	528813
Logistic Regression	Count(binary = True)	Unigram	79.08%	105473
	Count(binary = False)	Unigram	79.06%	105473
	Count(binary = True)	Ngrams = 1 - 2	**80.93%**	1119402
	TfIdf	Unigram	79.04%	105473
		Ngrams = 1 - 2	79.76%	1119402
		Ngrams = 1 - 2, minimum df = 2	**80.06%**	285502
		Ngrams = 1 - 3, minimum df = 2	79.95%	528963
SVM (with linear kernel or LinearSVC)	Count(binary = True)	Unigram	78.17%	92834
	Count(binary = False)	Unigram	78.00%	92834
	Count(binary = True)	Ngrams = 1 - 2	79.67%	1044111
	Tfidf	Unigram	78.64%	92834
		Ngram = 1 - 2	80.95%	1044702
		Ngram = 1 - 2 Minimum df = 2	80.88%	276548
		Ngram = 1 - 3 Minimum df = 2	**81.08%**	528642

Table 1: Accuracy scores of several combinations of vectorizors and features of our models

The results shown in Table 1 show that Logistic Regression and SVM both outperform MultinomialNB. Logistic Regression with count vectorizer (Ngrams = 1 - 2) performs the best with accuracy of **80.93%**, but SVM with TF-IDF vectorizer (Ngrams = 1 - 3, min df = 2) beats

Logistic Regression at **81.08%**. This variation of SVM reported a precision, recall and F-1 score of 81% as well.

The top features for the two best performing models above are quite consistent with what one would expect. Phrases/words describing negative emotions such as 'sad', 'not happy' and 'depressing' take the top places in most negative, and phrases such as 'can't wait' and 'no problem' are the top places in most positive. The top 5 for each side are shown in the table below:

	coef	feature
806989	-3.139758	sad
665911	-2.490865	not happy
252424	-2.444752	depressing
745485	-2.442341	poor
424393	-2.361783	headache
	coef	feature
667513	1.996444	not wait
664996	2.142285	not bad
171722	2.238859	ca wait
178295	2.267467	cant wait
660305	2.865570	no problem

	coef	feature
359674	-6.619423	sad
268406	-5.053097	miss
339391	-4.527159	poor
374114	-3.937091	sick
73756	-3.761326	ca
	coef	feature
298704	2.926092	no worri
299787	3.035872	not bad
78343	3.250267	cant wait
297970	3.428499	no problem
74763	4.285843	ca wait

Figure 4: Top features for Logistic Regression (left) and SVM (right)

N- grams:

When it comes to performance, unigrams give quite similar performance of roughly 76%, 79%, and 78% regardless of the vectorization method for MultinomialNB, LR, and SVM respectively. Bigrams add a significant improvement, since tweets saying "didn't love" or "not good" would be treated differently than just accounting for the individual words 'didn't' and 'love' or "not" and "good." By adding the use of both unigrams and bigrams, the model performance increases significantly, to 80.06% for LR and 80.95% for SVM (using tf-idf). Adding trigrams for SVM outperforms the rest of the variations at 81.08%, which is justified since the model now picks out phrases that make semantic sense with regard to sentiment.

Tfidf vs Countvectorizer:

Countvectorizer returns fits the model using all the unique ngrams in the document corpus. It transforms the tweets by creating a matrix which depends on a particular word's appearance in the tweet. With binary set to True, it will only have one or zero values, depending on whether or not the word occurs. Alternatively, with binary set to False, it will count the number of times that word occurs. Since tweets are quite small, they are therefore unlikely to contain the same word multiple times. For this reason, there are insignificant differences in the performance of the two approaches to count vectorizoration.

An alternative approach is the Term Frequency - Inverse Document Frequency representation. It compares the number of times the word appears in the tweet vs the number of times it appears in the document. In short, words that appear in many documents (like stop words such as is and has) are given scores close to zero, and words that appear only once in the text have scores close to 1.

In the logistic model, Td-Idf did not result in a very significant improvement in accuracy. However, it was particularly helpful in reducing the number of features due to a hyperparameter for minimum document frequency. Words appearing only once in the text (such as typos) can be removed entirely by setting minimum df to 2, without hindering the performance. In the above example, the number of features were nearly a quarter using this parameter when both unigrams and bigrams were used.

For the SVM model, Tf-idf reduced the number of features that were generated because of the minimum df feature, but the accuracy of the model also went up. This could be attributed to the fact that words that occur rarely in the dataset and can skew the classification algorithm were not considered when the minimum frequency was set to 2. Setting the min df to 3 did not result in any changes.

Conclusion:

We found that Logistic Regression (LR) and SVM perform better than baseline classification models like Multinomial Naive Bayes for our sentiment analysis classification task on Tweets. SVM slightly outperforms LR, with the best variation at 81.08%. This can be attributed to the SVM machine learning algorithm. Given the short length of texts and semantic significance 2-3-word phrases hold in tweets and text messages, the kernel transformation of the input data involved in SVMs allows for the model to distinguish between positive and negative feature representations better.

However, the results provided by LR are not very far off, and both the LR and SVM models can be further improved upon for better classification. One of the weaknesses our task had was that the existence of texts that are neither positive nor negative was not tackled.

This project could be further experimented and improved by trying different text processing techniques such as lemmatizing instead of stemming and increasing the list of stopwords currently being removed. Additionally, deep learning techniques involving neural networks (convolutional neural networks in particular because of the short length of text messages) can be investigated along with other vectorization methods such as word2vec models. To enhance the user experience with our project an improved UI design and implementation could be added.

Appendix:

Project Contributions:

Hassan: Made project.py file with implementation of GUI and high level structure. Did all work related to logistic regression. Inspected data provided. Researched on existing projects and completed the survey of related works' section.

Mahad: Formulated the high-level computational problem, researched on existing literature and datasets, and shortlisted machine learning models to be investigated given the classification task. Worked with Abdul to optimize the data preprocessing methods for better input to the classifiers and assisted him with nb.py. Did all work related to SVM. Ran experiments of all variations of the models (NB, LR, SVM), followed by an in-depth analysis of the results with Hassan and Abdul.

Abdul: Created textProcessing.py, textProcessing_methods.py and worked alongside Mahad and Hassan to improve the text preprocessing method in order to improve performance of models. Researched and performed experimentation to conclude which methods are best suited for our data cleaning needs. Worked on the Naive Bayes model with the help of Mahad. Completed the abstract, approaches section, Formulation of the Multinomial Naive Bayes Classifier in addition to contributing in other parts of the report. Ran experimentation for several combinations of vectorizors and features of our models.