Tweet Sentiment Analysis: Detecting Depression in Tweets

Andrew Liu Brandon Zhu Shahryar Kiani

Abstract

Depression is a significant mental health issue with widespread personal, social, and economic consequences. Symptoms such as persistent sadness, loss of interest, and behavioral changes can severely impact an individual's quality of life. In today's digital age, platforms like Twitter have become a space for individuals to share their emotions, offering an opportunity to detect early signs of depression. In this project, we use a dataset of over 20,000 tweets labeled as "depressed" or "non-depressed", and apply a Naive Bayes classifier to predict whether a user is depressed by looking at the set of their tweets. The text data is pre-processed by removing non-alphanumeric characters, punctuation, links, hashtags, and the model is trained to accurately distinguish between the two categories. By taking advantage of the high performance of Naive Bayes, the model is tested with many different feature spaces. Future work includes enhanced data pre-processing such as synonym replacement and stemming to enhance data quality, along with different classification techniques, such as Neural Networks or Multinomial Naive Bayes, to support early detection of mental health issues through social media analysis.

1 Introduction

Depression is a significant global concern, affecting millions of people and causing emotional, psychological, and social challenges. It can severely impact daily life, productivity, and overall well-being, making it a crucial issue to address. With the rise of social media, platforms like Twitter have become popular outlets for people to share their thoughts and feelings. This provides an opportunity for researchers to analyze public sentiment through sentiment analysis. Such analysis can offer insights into emotional trends, identify signs of depression, and help guide mental health support and interventions.

Detecting emotions from text, particularly in the context of depression and mental health, is challenging due to the nuanced nature of human emotions and how they are expressed. Extracting accurate emotional insights from textual data is complex, as language is inherently ambiguous, and emotional tones can vary greatly depending on context and individual expression. Over the years, many papers have explored different techniques to address these challenges and generalize across different texts. These methods range from traditional machine learning algorithms to more advanced neural networks and NLP techniques. One of

the original approaches to detecting depression using NLP was based on sentiment analysis, which focuses on classifying text as having a positive, negative, or neutral emotional tone. This simple classification was a stepping stone to more sophisticated methods that aim to categorize specific emotions, such as sadness, joy, or anger. Today, modern techniques incorporate models like convolutional neural networks, long short-term memory networks, and transformers, which can handle complex patterns in textual data.

Our project builds upon existing sentiment analysis research, aiming to go beyond a simple positive-negative scale. Understanding the nuances of human emotion is essential, and depression manifests in subtle ways that can be challenging to detect. By leveraging machine learning techniques trained on relevant datasets, our project seeks to contribute to the mental health field, helping individuals access necessary support while encouraging a societal understanding of depression's complexities. In summary, we experimented with various setups in our Naive Bayes model, including different configurations like n-grams and feature frequency requirements, to improve the accuracy of our text classification results.

2 Methods

Our main methodology for classifying tweets from depressed and non-depressed Twitter users is Naive Bayes for text classification. In the dataset that we used to train our Naive Bayes model, each user has a set of tweets associated with them (and an associated label for the user as depressed or not). To get the features that our Naive Bayes used, we took n-grams (1, 2, 3) from each tweet of each user. The purpose of this was to observe how looking at longer sequences of words as features would impact the accuracy of our model. One reason for looking at the sequence of words rather than at individual words on its own is that it maintains a better sense of the ordering of words, and captures words that change the meaning of a phrase (like "not"). We started by considering every n-gram in the dataset as a feature. Of course, this would result in a very large number of features, so we needed to trim out features that did not give us much information. We performed 2 main methods for feature trimming:

- Removing features that were frequent but insignificant to the classification
- Removing features that were rare in the dataset

One of the main techniques we used was removing features that were infrequent in the dataset. The cutoff point for how infrequent a feature should be be was a hyperparameter that we varied in our experiments. This allowed to us to search for a good cut-off point that prevented overfitting while also maximizing accuracy. There were also features that included cultural references and jargon used by various communities in Twitter not relevant to our study. We also noticed features that appeared very frequently but are insignificant to the training of our model. These features included stop words, a terminology used in NLP to refer to words that are commonly used in the English language, but are deemed insignificant. Such words include "a", "for", "the", etc., which only serve as a functional purpose to convey an idea. Since these features end up making it frequency cutoff, we added a separate filter

that detected features that only consisted of these words. By performing these operations, we were able to reduce our feature space and guide our model towards making more accurate classifications.

After we successfully extracted a desirable set of features from the dataset, we could then implement the Naive Bayes algorithm for text classification. We have two different approaches of training our Naive Bayes model for computing the probabilities of a user being depressed or non-depressed:

- Aggregating all the user's tweets into one large tweet and extracting the features from that to calculate the conditional probabilities.
- Extracting the features from each individual tweet of the users in the training set and using them to calculate the conditional probabilities.

The main difference between these two approaches comes from the treatment of repeated features. The second method gives more weight to features that are repeated across multiple tweets by a user, while the first method does not distinguish by the individual tweets. Instead, it simply looks at whether a feature appeared in any of the user's tweets. A feature that appeared in every tweet of a user and a feature that appeared in just one, would both count as appearing once for that user. For the first method, we simply used the number of depressed users as the divisor for the conditional probabilities for features in tweets users labelled depressed(and the same for non-depressed). For the second method, we used the total number of tweets by depressed users as the divisor. For reference, Figure 4 shows a bigram Naive Bayes Classifier using method 1, and Figure 2 shows a bigram classifier using method 2.

A different style of Naive Bayes classifier could have been used here. Multinomial Naive Bayes is a type of Naive Bayes classifier that is well suited to classifying text data. Specifically, the Multinomial Naive Bayes classifier does take into consideration the number of occurrences of a feature. Our implementation is a Bernoulli Naive Bayes classifier, which only looks at whether or not a feature is present. However, by training on an individual tweet, we were still able to include the frequency information in the training data in our classification decisions, albeit in an indirect way.

3 Data

We utilized a dataset from a Mental Health Twitter collection, featuring over 20,000 English tweets labeled to indicate the mental health status of the user, categorizing them as "depressed" or "non-depressed". This dataset is publicly available on Kaggle and was gathered using the Twitter API, providing real-world data on individuals' mental health states as expressed through tweets.

To prepare the data for classification, the following steps were taken:

- **Character Removal:** We removed non-alphanumeric characters such as hyperlinks, hashtags, and other symbols that do not contribute to the meaningful content of the tweets.

- **Special Cases:** Some tweets contained patterns or symbols specific to the Twitter platform, such as mentions (`@username`) and emojis. While emojis can convey emotions and affect sentiment analysis, they were excluded to streamline the dataset into purely textual content.

- **Data Splitting:** After cleaning, the dataset was split into training and test sets in an 80:20 ratio:

  – The dataset was split 80-20 by users, dividing it into training and test sets. However, since we were training on individual tweets, we then aggregated the depressed and non-depressed tweets from the users in the training set. This ensured we didn't train on tweets from users in the test set.

4 Experiments

In our experiment, we varied two hyperparameters of our Naive Bayes Classifier to see how the accuracy and precision changed over the test and training datasets in order to find the best setup for classifying the data. The first parameter that was varied was the length of the features that the Naive Bayes Classifier used. We used n-grams of length 1, 2, and 3. We expected that n-grams of length 2 would result in the best performance. This is because each feature would have more context since it represented a pair of words, which could put it ahead of unigrams, since they can't capture phrases like "not [word]". We also expected that trigrams wouldn't do as well, since they run into an issue with data sparsity, where there's a lower chance that an exact sequence of 3 words will be repeated. Essentially, with trigrams, it's likely our model would see more features in the test data that it didn't encounter in the training data.

The second hyperparameter that we varied was the minimum frequency cutoff for feature selection. N-grams that had a frequency over the dataset that was below the cutoff point would not be used as features. The minimum frequency cutoff was varied from 0 up to 975, 325, and 39, for unigram, bigram, and trigram based classifiers. The reason for different upper bounds comes from the fact that frequencies of N-grams decrease as N increases, since it is less likely a higher number of words are repeated in the same order.

We took advantage of the fact that training and testing a Naive Bayes classifier is relatively fast, which allowed us to run our classifier on a wide range of parameter configurations. This allowed us to find a set of configurations that maximized the performance of the classifier in an automated manner.

5 Results

Figures 1, 2, and 3 show the accuracy and precision of our unigram, bigram and trigram models on the test and training data. One of the first things that can be noticed in the data is the overfitting that is present when we don't cutoff any features. For all three of the

figures, we can see that the training accuracy and precision is 100%, while the accuracy and precision for the test data is significantly below that at the 0 Min Frequency Cutoff Point. Increasing the frequency cutoff point increases the test accuracy and precision for the models up to a point, after which they fall off. Overall, it seems that a simple frequency cutoff point can be an effective tool for improving the accuracy of a Naive Bayes Classifier and stopping overfitting.

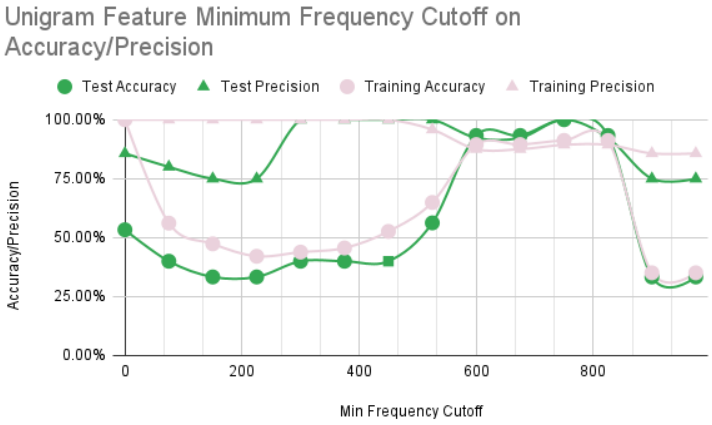

Figure 1: Unigram Model

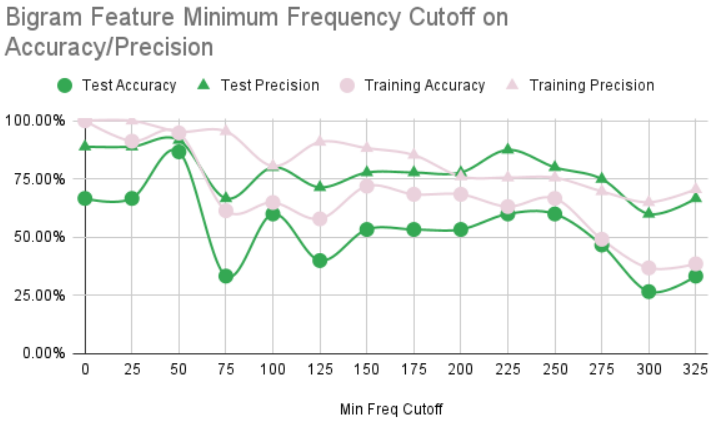

Figure 2: Bigram Model

However, although we were able to raise our accuracy by tuning the frequency cutoff point for all of the models, the underlying feature space did have a significant impact on the success of the models. The trigram model was not able to achieve a test accuracy higher than its training accuracy. This is likely due to the fact discussed above, where longer n-grams run into issues of data sparsity, where it's less likely that the same n-gram will reappear. In the case of our classifier, it's likely that trigram features that it used to make predictions with high accuracy on the training data didn't appear as often in the test data, lowering it's

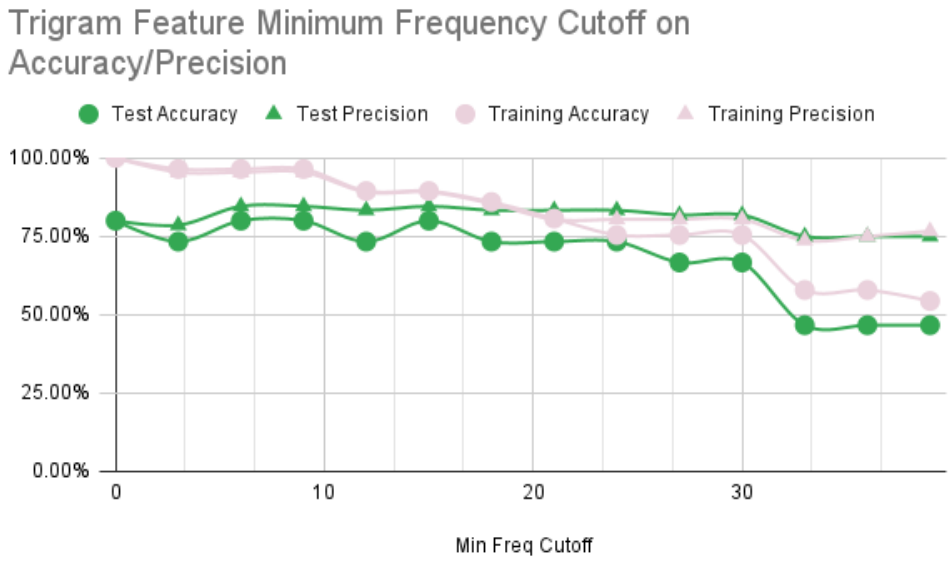

Figure 3: Trigram Model

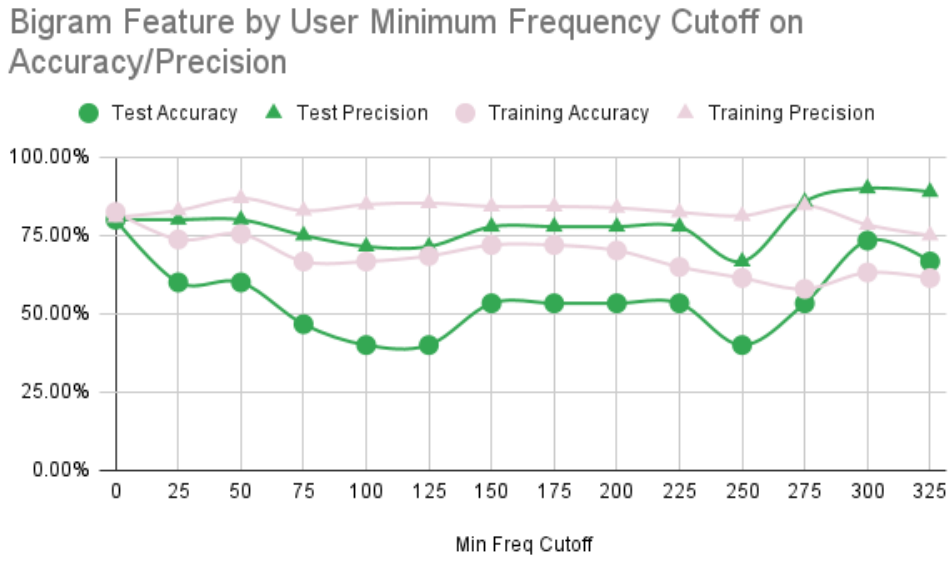

Figure 4: Method 1 Bigram Model

performance on the test data.

Overall, our experiments were able to find parameters that gave us Naive Bayes Classifiers with relatively high performance. Our Unigram model had 100% accuracy and precision on the test data along with 91.23% accuracy and 89.37% precision on the training data with a frequency cutoff of 750. Our Bigram model had 86.67% accuracy and 91.67% precision on the test data, along with 94.74% accuracy and 95.35% precision on the training data with a cutoff of 50.

6 Conclusion

Overall, our experiments show that properly tuned Naive Bayes Classifiers can have high accuracy and precision. By using code to test large combinations of hyperparameters, we were able to find configurations that maximized the performance of our models. We also learned the importance of adapting the classification approach to the structure of one's data. In our case, we found that trying to treat all the tweets of an individual user as a single tweet lost valuable information and decreased the accuracy of our model. By modifying the training phase of our classifier to look at the individual tweets of our user, we were able to guide our model towards a better understanding of the data, which helped it make much better predictions. Our experiments also demonstrated the importance of good feature selection through showing how reducing rare features can significantly reduce overfitting and improve performance on the test data.

In the future, some of the main experiments we'd like to complete are using different classification methods, such as Multinomial Naive Bayes or Neural Networks. We'd likely use a library for Multinomial Naive Bayes, since the main reason we didn't use it in this project was that we implemented Naive Bayes from scratch. Another area for possible future research is using improved data cleaning methods. During our research for this project, we discovered techniques such as stemming and synonym replacement, which help reduce the number of unique words and therefore reduce data sparsity. By using techniques like this, it's likely that we could gain additional performance from our existing Naive Bayes Classifier. One more future idea is collecting more data, our dataset consisted of only 20,000 tweets, meaning our test set was relatively small. We would like to see how our existing classifier would generalize to a larger test set. Overall, there are a lot of different avenues to extend the work in this project.

References

[**1**] Russell, Stuart, and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2010, https://people.engr.tamu.edu/guni/csce421/files/AI_Russell_Norvig.pdf.