Predicting Suicidality From Social Media Posts

Stephanie Yoshimoto

Abstract

Suicide can come suddenly, but by noticing patterns in an individual's posts on social media, one can be directed to the right help in time. To distinguish between a suicidal post versus a non-suicidal one, I have implemented two primary machine learning algorithms: a Naive Bayes classifier and an LSTM network. I evaluated both of these models by calculating precision, recall, and F1-score in addition to accuracy. Among the models, Naive Bayes performed slightly better than the LSTM, but both accuracies evaluated on the test set score 80% higher the baseline approach.

1. Introduction

Identifying suicidal thoughts and outward expressions early on when a person is depressed can spare the painful emotions that death leaves with family and friends.

The goal of this project is to better be able to predict if an individual will commit suicide based on social media posts. The input is a document (a post on Reddit) and the output is the classification suicide or non-suicide.

Based on the classification, sites such as Reddit and other online forums can use the result to direct individuals to suicide hotlines, psychiatric professionals, or, if this person has connected their contacts to the app, alert a parent or emergency contact. This project helps identify posts that are more indicative that the individual will attempt to commit suicide. The methods demonstrated here can be applied by content moderators to future social media posts that may not be clear whether they indicate suicide or non-suicide.

In this project, I have implemented a baseline, Naive Bayes, and a recurrent neural network, achieving an accuracy of approximately 90% on both Naive Bayes and the neural network. I cleaned the text input before running any machine learning methods by pre-processing the text so that different forms of a root word are used and that stopwords are removed. I chose to implement a recurrent neural network because it addresses a primary issue that Naive Bayes struggles with. As a result of these experiments, I have found that Naive Bayes and the neural network result in comparable accuracies on test sets. Either can be used to aid in making reliable predictions on social media posts.

2. Related Work

A group used demographic information for US counties to predict suicide rates. They used a regression based predictive model and the XGBoost regressor. Through their methods, they were able to decrease the amount of necessary features from 17 to 5 (population, % white population, median age, % African American population, and % female population). Though their work in this study may be incomplete, as the authors acknowledge that there may be suicides that were not reported, or suicides that were meant to cause harm but not death. Additionally, there was no demographic information on % tribal population, which may or may not have been a significant factor for this study (Kumar et al., 2022).

A similar experiment was done a year earlier, where instead, researchers looked at features such as substance use, suicidal thoughts, and academic pressures instead of demographics to evaluate suicidality in college students. They used a random forests model to find that the most influential factors were 12-month suicidal thoughts, trait anxiety, depression symptoms, and self-esteem. However, since their data came from questionnaires filled out by college students, they cannot prove that all students were answering entirely truthfully for the initial or follow-up responses. They acknowledge that they should have collected data from more sources (Macalli et al., 2021).

Another project from 2021 tried various models on classifying suicidality from Twitter posts. They used k-Nearest Neighbors, support vector machines, and a form of attention called C-attention. It was found that C-attention was the most effective in long-range predictions whereas the other two methods were better at short-range predictions. In their process, they chose both hand-selected features and latent features based on word embeddings. Some methods performed better for hand-selected features while others were better with latent features. They arrived at relatively low values for F1-score and AUC, leading them to believe that they overfit their model, ultimately agreeing that they needed to find a way to better distinguish what features to use and how to choose them.

My approach focuses on social media posts written by suicidal and non-suicidal individuals, not on features such as demographic information, though my goal aligns with all the works. I decided to use deep learning like in the last approach to be able to capture relationships between words, but by instead implementing a version of a recurrent neural network that can handle long-range dependencies. I made sure to use early stopping so I would not overfit the model, and ended up with a better F1-score than the last method. If the last example had used early stopping or weight decay to handle overfitting, their F1-score could have been higher (Wang et al., 2021).

3. Dataset and Evaluation

I am using the Suicide and Depression Detection dataset from Kaggle (Komati, 2021). It contains over 232,000 binary classification examples drawn from the "SuicideWatch" and "depression" sub-Reddits from the social media platform, Reddit. Each example contains the classification non-suicide or suicide, as well as the text of a Reddit post. Out of the total 232,072 examples, there are 116,037 suicide examples and 116,035 non-suicide examples. The following is an example of a suicide post: "What is the best way to do it?I'm not looking to be talked out of it. What would be the most effective, easiest way to go?". The posts can vary in length from one sentence long to multiple paragraphs of text. The average word count is 131.5 words.

The dataset has been shuffled randomly and then split into training, development, and test sets with 70% training, 10% development, and 20% test. This corresponds to roughly 162,400 examples for training, 23,000 examples for dev, and 46,000 examples for test. Each set contains a variety of non-suicide and suicide examples to ensure coverage of both classes.

The models have been evaluated based on accuracy, recall, precision, and F1-score. I used a confusion matrix to generate the inputs for the metrics. These evaluation tools help to determine how consistently the model makes accurate classifications. I used precision to measure how consistently correct positive predictions are made out of all positive predictions, recall to evaluate how well the model can detect the suicide class, and F1-score to combine the two. Precision helps to identify any false positives, so as to prevent taking unneeded measures to prevent suicide if a person is not suicidal. Recall helps to evaluate how well the positive, suicide class is predicted, so I would want high recall to identify as many suicidal posts as possible so that proper action can be taken.

4. Methods

4.1. Majority-Rule Baseline

I first used a majority-rule baseline in which I calculated the most common occurrence in {non-suicide, suicide} in the training set, and assigned the label to every example in the test set. In other words, I found the maximum of $\sum_{i=1}^{n} \mathbb{1}(y^{(i)} = \text{suicide})$ and $\sum_{i=1}^{n} \mathbb{1}(y(i) = \text{non-suicide})$, and applied the most common label to every example in the test set. I divided the training set and test set into a random 80%-20% split, since there was no need to test hyperparameters with a development set.

4.2. Naive Bayes

4.2.1. Equations

I implemented Naive Bayes to find the most likely class for an example according to the equation

$$P(Y \mid X) = \frac{P(Y) * P(X \mid Y)}{P(X)}$$

where Y represents either the classification suicide or non-suicide and X represents the input, which is a Reddit post. The maximum value was found between

$$P(\text{suicide} \mid X) = P(\text{suicide}) * P(X \mid \text{suicide})$$

and

$$P(\text{non-suicide} \mid X) = P(\text{non-suicide}) * P(X \mid \text{non-suicide})$$

Since $P(X)$ is the same in both cases, it can be ignored. $P(X \mid Y)$ was broken down using the Naive Bayes assumption of conditional independence,

$$P(X \mid Y) = \prod_{i=1}^{n} P(X_i \mid Y)$$

where X_i represents one word in a document. The full equation was transformed into a sum of logs

$$P(Y \mid X) = \log P(Y) + \sum_{i=1}^{n} \log P(X_i \mid Y)$$

to ensure no multiplication of small values.

To avoid the possibility than an unseen word from the training set might come up in the test set, Laplace smoothing was added with $\lambda = .075$. This value was found to give the highest accuracy after training the model and then evaluating multiple models on the development set.

4.2.2. Text Pre-Processing

The features of one input X are represented by the words in the input document. These features were cleaned to ensure

uniformity among the data so that similarly formed words would be treated as the same words. I pre-processed the text by converting all Reddit posts to lowercase and removing punctuation, including commas, periods, and contractions like in the words "don't" and "shouldn't". I then filtered out stopwords that are common to both classifications and therefore unnecessary in distinguishing a class, such as articles and pronouns. The Reddit posts were then lemmatized to get the root word, so that "books" and "book" are treated as the same word.

4.3. Recurrent Neural Network

4.3.1. Word Embeddings

I used a set of approximately 1 million pretrained word vectors each of dimension 300, retrieved from FastText (Mikolov et al., 2018). Each line in the file contained the word followed by 300 space-separated floating point numbers representing the word vector.

For each batch, before running the forward pass of the neural network, I mapped every word in the input Reddit post to the index of the word in the file. Since the input would be unequal in size based on the length of the posts, I padded the shorter sentences with zeros at the end.

4.3.2. Architecture

I used an LSTM rather than a traditional RNN because of its ability to handle long-range dependencies. For this task, I used Pytorch's built-in LSTM model. The LSTM manages the hidden state and the cell state, which is responsible for long-term memory. The Pytorch model uses three gates, which control what comes in and out of the network. The forget gate determines how much information will be retained from the previous step, the input gate determines if the cell state will be updated, and the output gate determines the value of the new hidden state (Dolphin, 2020). LSTMs require a hidden state and a cell state when initialized; both of these were initialized to zero tensors at the beginning of every epoch. After the initialization, each batch would go through the forward pass of the network as follows:

- Embed the input with the pretrained word embeddings
- Apply an LSTM layer to map from the input dimension of 300 to `hidden_dim`
- Apply dropout to the output of the LSTM
- Use a hidden layer to map from `hidden_dim` to `NUM_CLASSES`

4.3.3. Model Settings

During training, I used Adam as an optimizer, and iterated over the data in batches. I decided to tune batch size, dropout probability, learning rate, and `hidden_dim`. These were chosen because they were seen to sufficiently affect the model upon multiple runs. The number of epochs to run for did not matter much, since all accuracies could be determined within 20 epochs.

In order to prevent overfitting, I utilized early stopping by selecting the best model evaluated on dev to evaluate the test set on.

4.4. Model Comparison

Using a recurrent neural network has its advantages over Naive Bayes and majority-rule baselines. Recurrent neural networks—or in this case, LSTM networks—are meant to handle word order so that they recognize the relationship between words, not simply count the frequency of one word or another or assign a class based on a majority-rule. In this way, it is better able to interpret the meaning of a sentence and how it relates to previous and following sentences in a post, capturing the intent of the message as a whole.

5. Experiments

Each time I ran the majority-rule baseline, I received an accuracy around 49-50%. When running this baseline multiple times, the majority class would switch from non-suicide to suicide or vice-versa based on how the data was split after a random shuffle. The output of one run is shown in the graph below. Here, the model found that suicide was the most common among training examples, so it predicted suicide for every test example. This led to a 49.793% accuracy on the test set.

Figure 1. Majority-Rule Baseline Confusion Matrix Evaluated on Test Set

When running Naive Bayes, I got a training accuracy of 90.845%. I ran the model multiple times on the development set to find an appropriate λ and got the following results.

Table 1. Lambda hyperparameter evaluated for Naive Bayes.

λ	DEV SET ACCURACY
10	79.664%
1	89.871%
0.5	90.491%
0.1	90.914%
0.075	**90.918%**
0.05	90.901%
0.01	90.836%
0.001	90.616%

From this, I found that the choice λ should be 0.075. I ran Naive Bayes on the test set and got an accuracy of 90.549%.

Naive Bayes far outperformed the majority-rule baseline. The majority-rule baseline only assigned a class to the test examples based on training data, ignoring the structure and content of the test examples. Naive Bayes, on the other hand, determined the best classification based on frequency counts of words in each test example and assigned a label based on probabilistic outcome.

When running experiments on my LSTM network, I found that training would take significantly more time than Naive Bayes. After trying a few epochs of the full training dataset and comparing this against running with around 16,000 examples, I did not see a noticeable difference in accuracy. Because of this, I decided to run the LSTM with 16,384 training examples, 2,304 examples in the dev set, and 3,840 examples in the test set.

I was able to find the best hyperparameters by running various values on the dev set. When evaluating a hyperparameter, I kept all other hyperparameters constant.

Table 2. Batch size hyperparameter evaluated for LSTM.

BATCH SIZE	DEV SET ACCURACY
32	81.901%
64	**89.714%**
128	88.845%

Table 3. Dropout hyperparameter evaluated for LSTM.

P	DEV SET ACCURACY
0.0	86.372%
0.1	**89.236%**
0.2	83.420%
0.3	85.200%

Table 4. Learning rate hyperparameter evaluated for LSTM.

η	DEV SET ACCURACY
1E-3	88.932%
1E-2	**89.714%**
1E-1	89.236%
1	66.623%

Table 5. Hidden nodes hyperparameter evaluated for LSTM.

NODES	DEV SET ACCURACY
100	77.517%
200	**89.714%**
300	70.095%

After experimenting with hyperparameters, I found that the best options are batch_size = 64, dropout = 0.1, η = 1e-2, and `hidden_dim` = 200. I used these results and got a train accuracy of 89.673%, a dev set accuracy of 89.106%, and a test set accuracy of 89.323%.

The confusion matrix for the LSTM is:

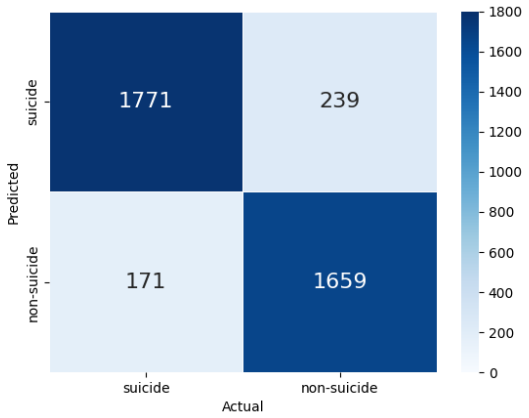

Figure 2. LSTM Confusion Matrix Evaluated on Test Set

The model is performing at nearly twice the accuracy of the majority-rule baseline, similar to Naive Bayes, since it is able to assess posts based on what words the post contains rather than a majority-rule of the training data.

Naive Bayes managed to outperform the LSTM model in this case, though the difference in accuracy is small. This may be caused by slightly lower accuracy when running on a subset of data, which could have been resolved with more data. A smaller dataset also means it is more sensitive to hyperparameters such as dropout or the amount

of hidden nodes. The accuracy could be attributed to the relatively simple architecture of the neural network. I could have achieved higher accuracy by trying techniques such as batch normalization or adjusting the parameters of the Adam optimizer.

I was surprised that a neural network could not produce significantly better results than a linear classifier. It is likely that since the input comes from social media, the authors of the posts may express themselves more dramatically in order to gain attention or empathy. For instance, a post such as "My mom just grounded me! I can't go to prom! I want to die!" is overly dramatic. It's quite often that someone would exaggerate their emotions when posting behind the safety of a screen, so the phrase "I want to die" can be taken in multiple ways, whether someone is being sarcastic, over-dramatic, or legitimately depressed and suicidal. While recurrent neural networks can account well for word order, they may not be as adept at detecting tone, something that is very prevalent and can vary widely in social media.

6. Discussion

Accuracy, precision, recall, and F1-score for the majority-rule baseline are calculated according to the following equations:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} = .49793 \quad (1)$$

$$\text{Precision} = \frac{TP}{TP + FP} = .49793 \quad (2)$$

$$\text{Recall} = \frac{TP}{TP + FN} = 1.0 \quad (3)$$

$$\text{F1-score} = 2(\frac{\text{PR}}{\text{P+R}}) = .664828 \quad (4)$$

The confusion matrix outputted for Naive Bayes is:

actual\predicted	suicide	non-suicide
suicide	22300	910
non-suicide	3482	19780

The evaluation metrics for Naive Bayes are as follows:

$$\text{Accuracy} = .9055 \quad (5)$$

$$\text{Precision} = .8649 \quad (6)$$

$$\text{Recall} = .9608 \quad (7)$$

$$\text{F1-score} = .91 \quad (8)$$

Naive Bayes performed well as a binary classifier, achieving around 90% accuracy and a high F1-score. The model was able to pick out important words that contributed to the suicide class and apply it to test data. A high frequency of words associated with the suicide label made the text more likely to classify as suicidal. Since language follows a similar pattern, where words expressing suicidal thoughts come out in suicidal posts, it makes Naive Bayes a good classifier for this problem.

However, there were a number of examples that were not classified correctly. Looking at the data for what was incorrectly classified, I see that examples that were non-suicide but classified as suicide include relationship subjects such as "wife", "girlfriend", and "boyfriend". Since common causes of suicide are close, interpersonal relationships, the model may have attributed relationship-related words to suicide.

It is also possible that suicide examples that are misclassified as non-suicide may include a long background story, and the words in this story may outweigh suicidal words later on in the text. This is an example of when Naive Bayes' ignoring of word order causes misclassifications. Another way in which Naive Bayes' use of frequency counts could have caused issues is from synonyms. Words such as "mad" and "angry", which convey the same emotion, nevertheless could be weighted differently based on how much they occur in the examples.

However, word vectors solve this issue, since it takes advantage of the idea that similar words in similar contexts should have similar word vector representations. Using an LSTM helped achieve even weighting across synonyms.

When running LSTM on the hyperparameters of batch_size = 64, dropout = 0.1, η = 1e-2, and `hidden_dim` = 200, the resulting confusion matrix and evaluation metrics were:

actual\predicted	suicide	non-suicide
suicide	1771	171
non-suicide	239	1659

$$\text{Accuracy} = .8932 \quad (9)$$

$$\text{Precision} = .8811 \quad (10)$$

$$\text{Recall} = .9119 \quad (11)$$

$$\text{F1-score} = .8963 \quad (12)$$

Just as with Naive Bayes, accuracy and F1-score are close to 1.0. From this, I know that the network was successfully learning features and learning how to classify what a suicide post looked like versus a non-suicide post. Because of its deep learning capabilities, I would expect the network to achieve higher accuracy, but given some implementation choices such as the settings of the model and the optimizer and the architecture of the network, higher accuracy likely would not have been achievable without substantial changes.

With text input, I had to make a decision about how to handle typos. I handled this case by ignoring typo words completely. Looking at posts that were misclassified, I see

that most of them contain typos. This could be a reason for low accuracy, since important words or phrases could have been left out in the process of mapping words to the indices in the vocabulary.

The post "tonight nightim exited" contained a typo and was classified as non-suicide even though it was a suicide post. This does not clearly imply suicide, so it makes sense that the model misinterpreted the meaning of the sentence. A reason it could have been suicidal when posted is because the author might have been excited or relieved to end their suffering that night. This wording would not typically be found in a suicide post, which would mainly contain words of anger, hurt, or frustration. Ensuingly, the classification of non-suicide makes sense. It is also possible that this post did not provide enough context about the author's situation or backstory, so it was hard for the model to discern a classification.

7. Conclusion

In this project, I implemented a majority-rule baseline, then used Naive Bayes and an LSTM to improve upon the task of text classification. I found that Naive Bayes outperforms the baseline with an accuracy that is almost twice as high, and the LSTM performs similarly.

The most challenging part of this project was writing the code for the LSTM, when I had to figure out the training loop of the neural network. I had to determine how to process the million word vectors and transform the input text to indices in the vocabulary, both in ways that would not use up too much RAM. Additionally, since a neural network is much more complex than linear models, it took much longer to train. This meant that testing hyperparameters took up most of the time during my experiments for the LSTM.

It was surprising to me that the LSTM performed roughly the same as Naive Bayes. I can attribute this to the relatively straightforward architecture of the network. Now that I have a working implementation of an LSTM, though, to improve upon this classification task in the future, I would want to try out other parameters of Adam and add more complexity to the network so that it has more capability to learn text features. Ultimately, I'll have to take into account the many nuances of text input. Elements such as tone and converting typos into valid input are imperative to building a truly effective text classifier.

8. Code & Data Submission

The code and dataset for this project are available at: Google Drive (requires USC login).

References

Dolphin, R. Lstm networks: A detailed explanation, Oct 2020. URL https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9

Komati, N. Kaggle. https://www.kaggle.com/datasets/nikhileswarkomati/suicide-watch?resource=download, 2021.

Kumar, V., Sznajder, K. K., and Kumara, S. Machine learning based suicide prediction and development of suicide vulnerability index for us counties, Jun 2022. URL https://www.nature.com/articles/s44184-022-00002-x#citeas.

Macalli, M., Navarro, M., Orri, M., Tournier, M., Thiébaut, R., Côté, S. M., and Tzourio, C. A machine learning approach for predicting suicidal thoughts and behaviours among college students, Jun 2021. URL https://www.nature.com/articles/s41598-021-90728-z.

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. Advances in pre-training distributed word representations. In Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018), 2018.

Wang, N., Luo, F., Shivtare, Y., Badal, V., Subbalakshimi, K. P., Chandramouli, R., and Lee, E. Learning models for suicide prediction from social media posts, 2021. URL https://aclanthology.org/2021.clpsych-1.9.pdf.