Detecting Depression Through Tweets

Diveesh Singh
Department of Computer Science
Stanford University
Stanford, CA 94304
diveesh@stanford.edu

Aileen Wang
Department of Computer Science
Stanford University
Stanford, CA 94304
aileen15@stanford.edu

Abstract

This paper aims to predict depression given a Twitter user's tweet(s). We create our own dataset by scraping tweets off various Twitter pages, and label them with the aid of the polarity score generated from Textblob's python package. Then, we construct several deep learning models (RNN, GRU, and CNN) to generate predictions on this dataset. For these models, we examine the effects of character-based vs. word-based models, and pretrained embeddings vs. learned embeddings. We find that the best-performing models are word-based GRU, with 98% accuracy, and word-based CNN, with 97 % accuracy.

1 Introduction

1.1 Motivation

Analyzing the content of Tweets has become an increasingly more popular method to understand and make predictions about human social behaviors. Given the frequency with which Twitter is used by the broad population, it is a very rich source of data that can be used to analyze a variety of these behaviors. For this project, we chose to focus our efforts on how apparent is it from an individual's Tweets that they are suffering from depression. Depression is a significant subject of interest since it is a mental illness that adversely affects a large part of the world population (350 million people worldwide), and is often associated with other mental disorders. Depression itself is a multi-faceted illness, and therefore it affects different people in different ways and to different extents.

Detecting depression through NLP is a more convoluted task than simple sentiment analysis, since the labels "not-depressed/depressed" cannot be equated with the labels "happy/sad". Furthermore, indications of depression in tweets are often subtle, and thus not immediately obvious to the human reader. These subtle indications, however, may be reflected in the nuances of someone's language, which we predict can be captured by a variety of deep learning methods. By using a wide set of examples, along with state-of-the-art NLP techniques, we hope to create a robust model that is sensitive to the variations of depression on an individual basis.

The findings of this project will be useful in predicting depression in individuals, even if they are unwilling to discuss their issues with a professional. Health professionals will be more aware of which specific demographics are more affected by depression, and potentially create depression awareness/prevention messages to help those demographics.

1.2 Problem Definition

Our project aims to do the following:

- Generate labeled tweets dataset from scratch

- Construct neural-based architectures (RNN, GRU, CNN) to predict depression
- Tune model parameters to optimize performance
- Analyze performance of character-based vs. word-based models
- Analyze performance of pre-trained embeddings vs. learned embeddings

2 Background and Previous Work

There has been a lot of previous work done on sentiment analysis of Tweets using a variety of standard NLP techniques. Specifically a lot of these papers discussed the concept of generating embeddings for words and/or characters, and then feeding those embeddings into the model for training purposes. One paper, [4], talks about using a deep convolutional network to do sentiment analysis of short texts i.e. Tweets. The main issue in this type of analysis is that many Tweets lack context, and therefore require some combination of text analysis and prior knowledge to be able to effectively classify a piece of text. First, it extracts word and character level embeddings for all the Tweets; with this information, it then trains convolutional layers to generate sentence level features by analyzing all the windows of the sentence. This vector of sentence level features is passed through 2 normal neural network layers, and the output of the final layer is used as the prediction. This paper was useful in formulating our approach because it discussed effective ways of using character embeddings in generating sentence-level embeddings.

There were some papers that took a more creative approach to solving the problem of Tweet analysis. Specifically, a paper from MIT [6] coins the term *Tweet2Vec*, as a technique to learn Tweet embeddings. It learns these embeddings via a character-level CNN-LSTM encoder-decoder. At its core, the way it encodes the data is using a series of convolutional layers to extract features at the character level, and then passing these features sequentially through an LSTM layer. To decode the data, it passes the data through two LSTM layers, and the output could be used to predict the next character in the sequence.

3 Dataset

Our dataset consisted of 13,385 tweets scraped from various Twitter pages, where each tweet is labeled with 1 or 0; 1 indicates the user who posted the tweet suffers from depression, 0 indicates the user who posted the tweet does not suffer from depression. The train set (train.csv with 10,708 tweets) and dev set (dev.csv with 2,677 tweets) are randomly shuffled and selected from the above dataset.

3.1 Collecting the Dataset

We collected the dataset using a few Twitter scraper utilities we found online, including [5]. All scraped tweets are originally unlabeled.

To collect potentially positive examples, we searched for pages on Twitter that had something to do with depression; specifically, we would require that the word ”depression” was somewhere in the user name or title of the page. Examples like ”Depression Quotes”, ”Depression Notes”, and ”Damn Depression” were all useful pages in our dataset collection. We made sure not to scrape Tweets from pages that were more geared towards depression awareness, or those pages managed by organizations that are working to help people with depression, as a lot of these pages were not owned by users who were potentially depressed. After inspecting the content of the pages we had shortlisted as ”valid” pages, we collected their corresponding tweets and compiled them into a list of potentially positive examples.

For potentially negative examples, we pulled examples from a wide range of pages (the news, sports, dance teams etc.); we also made sure to get pages that had to do with other emotions (anger, happiness, etc.), so that our model could be able to distinguish between emotions due to depression and other emotions.

After collecting the tweets, we labeled them manually; to aid our labeling, we wrote a script that uses the Textblob python package to calculate a polarity score for each tweet; based on these scores, the script labels each tweet as 1 for depressed, 0 for nondepressed. After we run this script, we manually filter through all the tweets to check if the predictions made by the Textblob algorithm are reasonable, and update the labels accordingly.
An example "depressed" tweet looked like this:

> it sucks, doesn't it? feeling like you're not good enough, no matter how hard you try.

3.2 Cleaning the Dataset

3.2.1 Filtering Irrelevant Examples

Many of the examples in the initial dataset were not actually relevant for our use case. This is because several of the examples had something like "Here's a new YouTube post!" or "Account got suspended for a while, but now we're back!" that did not seem to indicate anything about the user, their thoughts, or their situation. We filtered our entire dataset to take out these arbitrary examples.

3.2.2 Sanitizing Individual Examples

For the remaining examples, we sanitized each tweet so that they did not contain irrelevant text, so they would be suitable input for our various models. One category of irrelevant text, for both word-based and character-based models, are hyperlinks, because they do not add much to the actual content of the tweet. On top of that, hyperlinks do not have a corresponding word embedding from Word2Vec or GLoVe, nor could we have generated useful word level embeddings for these hyperlinks. Another category of irrelevant text, for word-based models only, is hashtags and miscellaneous symbols, because similarly to hyperlinks, they do not have corresponding word embeddings.

4 Technical Approaches

4.1 Baseline: Logistic Regression and SVM

For our baseline algorithm, we used scikitlearn logistic regression with the following parameters

- Regularization $C = 0.1$ and Solver = 'lbfgs'.

and SVM classifiers with the following parameters

- Kernel="rbf"and Penalty for the error term $C = 0.025$

to perform binary classification. First, for each tweet in the dataset, we use the Word2Vec function from the gensim python library to to create word embeddings for each word within the tweet, represented as a 100 dimension feature vector. Then, we get the feature vector for each tweet by averaging all the words vectors within the tweet. Finally, we apply our baseline classifiers to the collection of tweet feature vectors and the corresponding labels to train and test our baseline models.

4.2 Word-based RNN Model

Once we had established our baseline, the first model we implemented was a simple RNN model. RNNs are popular for tackling NLP tasks because they process text in a sequential manner, and they can take in any length input.

We preprocessed the train.csv and dev.csv files, so the input to the RNN model was a list of (tweet, label) pairings. Each tweet was represented as a vector of words, where each word is represented with a discrete ID that can be used to look up its Word2Vec word embedding. Each tweet was padded to a maximum length of 30 words.

For each timestep t of the RNN, we fed in a batch of words that correspond to the t^{th} word

for each tweet in the current batch. The final hidden state, h, was then fed through another neural network layer, followed by the softmax function, from which the prediction was made. We computed the cross-entropy loss between the predicted labels and the true labels. We used the Adam optimizer during training to minimize this loss as follows:

$$\begin{aligned} h^{(t)} &= W_t h^{(t-1)} + W_x x_t + b_1 \\ o &= U h^{(t_{final})} + b_2 \\ softmax &= \frac{e^o}{\sum_{j=1}^{n} e^j} \\ CE &= \sum_{i=1}^{n} y_i log(\hat{y}_i) \end{aligned}$$

4.3 Word-based GRU Model

Our next update to the model was changing the basic cell unit from an RNN cell to a GRU cell, while still using word-level embeddings. Using GRU cells are more powerful than using vanilla RNN cells because they incorporate two additional gates (update gate and reset gate), which gives the model more flexibility in processing input; GRUs can create of shortcut connections between text separated by an arbitrary number of timesteps, allowing the model's memory to capture more long-term dependencies in text. GRU cells are computationally more efficient than their variant, LSTM (long short term memory) cells.

Outside of cell type, everything else, from preprocessing the data to training/evaluation techniques remained the same as the simple RNN model. The GRU cell model works as follows:

$$\begin{aligned} \text{update gate: } z &= \sigma(x_t U_z + h^{(t-1)} W_z) \\ \text{reset gate: } r &= \sigma(x_t U_r + h^{(t-1)} W_r) \\ \text{new memory: } \tilde{h}^{(t)} &= tanh(x_t U_h + (h^{(t-1)} \cdot r) W_h) \\ \text{final memory: } h^{(t)} &= (1 - z) \cdot \tilde{h}^{(t)} + z \cdot h^{(t-1)} \end{aligned}$$

4.4 Character-based GRU Model

In addition training models that used word embeddings, we wanted to explore the effectiveness of using character level embeddings. Several papers had suggested that using character level embeddings, especially for content like Tweets, actually capture more cases where users repeat letters to indicate emphasis (e.g "hellllooooOOo"), and also takes into account the fact that a hashtag can indicate something important about the sentence; our word-based model had cleaned out hashtags, but this character, among other characters, was used in this character-based model.

The character embeddings (for the normal ASCII range) were derived from the GloVE word embeddings, as inspired by [3]. We calculate the embedding for a specific character as follows: Every time the character appears, we infer its embedding from the parent embedding. We add all these "inferred" character embeddings together across the whole corpus, and divide by the character's frequency, resulting in the final character embedding. Figure 1 shows the character embeddings, when put into a tSNE visualization.

Similarly to the word-based RNN model and GRU model, we preprocessed the train.csv and dev.csv files, so the input to the RNN is a list of (tweet, label) pairings. However, each tweet was now represented as a vector of characters, where each character was represented with a discrete ID that can be used to look up its character embedding. Each tweet was padded to a maximum length of 150 characters. Training and evaluation techniques remained the same as the word-based RNN model and GRU model.

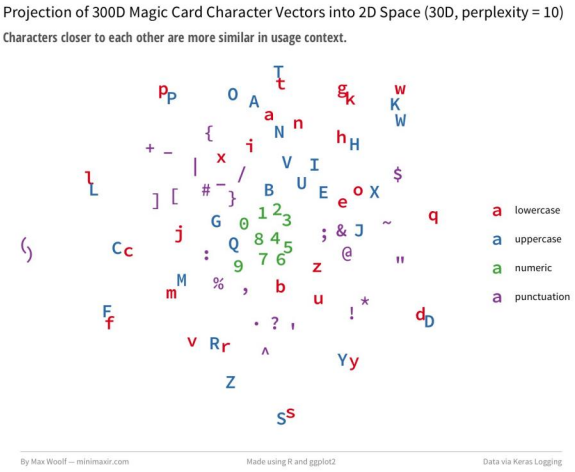


Figure 1: Character embedding tSNE Visualization (from [3])

4.5 Hyperparameters

For the above 3 RNN-based models (basic RNN with words, GRU with words, GRU with characaters) we used the following hyperparameters: Learning Rate: .001, Epochs: 10, Dropout Probability: 0.5, Minibatch Size: 32, Hidden State Size: 300.

4.6 Using a Word-based CNN Model to Generate Sentence Embeddings

The final model that we built was a convolutional neural network, inspired by [1] and [2]. Although they are most common in computer vision, CNNs have recently been applied to NLP tasks and have shown promising results in text classification.

We preprocess the train.csv and dev.csv files similarly to the word-based RNN and GRU models. A major difference between this CNN and other RNN-based models are that this model learns the embedding matrix W at training time, while RNN-based models used pretrained embeddings.

Figure 2 shows a simple visualization of the CNN architecture. We use different filter sizes (3, 4, 5) to slide over 3, 4, 5 words at a time, respectively. For each filter size, we calculate the outputs of the convolution layer, and apply the ReLU nonlinearity and max-pooling on these outputs. We combine all these pooled features from each filter size to form our feature vector on the final layer. We add a dropout layer on this feature vector with a probability of 0.5, and use cross entropy to calculate the loss.

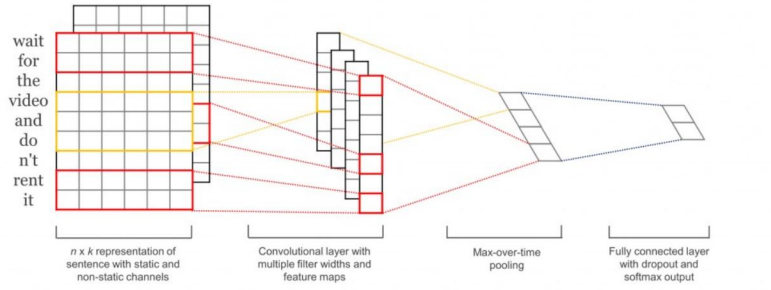


Figure 2: Visualization of CNN (from [2])

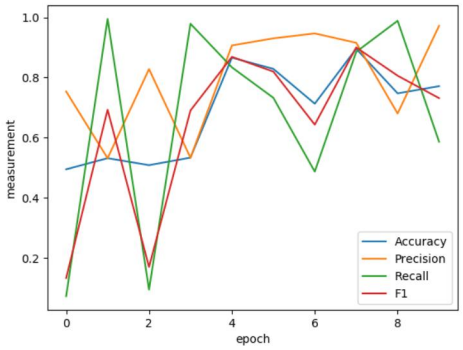
5 Experiments, Results and Analysis

5.1 Quantitative Analysis

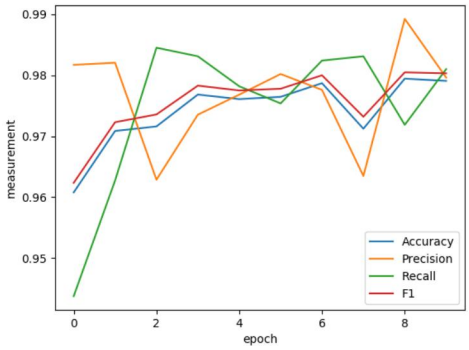
To quantitatively evaluate our models, we mainly used Accuracy(Acc), Precision(P), Recall(R), F1 score, and confusion matrix (CM) defined as follows.

$$Acc = \frac{tp + tn}{tp + tn + fp + fn}$$
$$P = \frac{tp}{tp + fp}$$
$$R = \frac{tp}{tp + fn}$$
$$F1 = \frac{2 \times P \times R}{P + R}$$
$$CM = \begin{bmatrix} tn & fp \\ fn & tp \end{bmatrix}$$

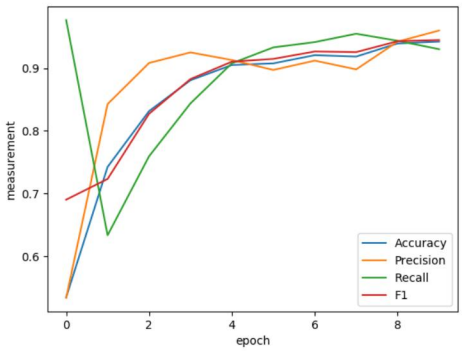
Where tp, tn, fp, fn are the number of true positive, true negative, false positive, and false negative respectively. In addition, We measured training accuracy and validation accuracy throughout the training process for each of the models, along with the training loss and validation loss.



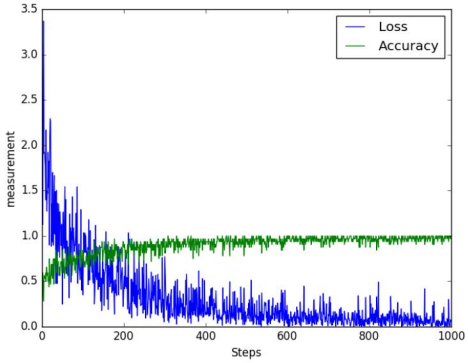
(a) Depression prediction using RNN



(b) Depression prediction using GRU



(c) Depression prediction using GRU with CHAR



(d) Depression prediction using CNN

Figure 3: (a) - (c): Measurements of Accuracy, Precision, Recall, and F1 score. (d) Loss and Train Accuracy

Label	Acc	P	R	F1
Nondepressed	0.90	0.87	0.91	0.89
Depressed	0.90	0.92	0.89	0.90

(a)

Label	Acc	P	R	F1
Nondepressed	0.98	0.97	0.99	0.98
Depressed	0.98	0.99	0.97	0.98

(b)

Label	Acc	P	R	F1
Nondepressed	0.94	0.92	0.96	0.94
Depressed	0.94	0.96	0.93	0.94

(c)

Label	Acc	P	R	F1
Nondepressed	0.97	0.97	0.98	0.98
Depressed	0.98	0.99	0.98	0.99

(d)

Table 1: Best Measurements for (a) RNN, (b) GRU, (c) GRU with CHAR, (d) CNN.

5.2 Qualitative Analysis

5.2.1 Analysis of Basic RNN vs. GRU on Word Embeddings

Using a GRU cell performed substantially better than using a basic RNN cell when we used word embeddings. This is likely due to the fact that using an RNN cell may bias the model into paying more attention to the words that appear at the very end of the sentence, whereas the GRU cell allows different parts of the sentence to play an active role in deciding the final prediction. A specific tweet we saw that illustrated this trend well was:

- `Everything sucks. Why do good things always happen to other people?`

In this example, it is clear that the beginning of the tweet is showing some form of frustration, and is the main indicator in classifying the sentence. The second part of the sentence, while in context of the first two words, does add some potentially indicative content, but when the RNN cell only takes the end part of the sentence into account, it almost ignores the first part. Likely due to this reasoning, the Tweet was misclassified by the RNN cell version, but not by the GRU cell version.

5.2.2 Advantages/Disadvantages of Using Character-Based Embeddings

From Figure 3, we compare the performance of word-based GRU (b) and character-based GRU (c). Although the performance is slightly lower, we can see that using character embeddings, results in significantly smoother curves for accuracy, precision, and F1. This is likely due to the fact that tweets tend to be more idiosyncratic, depending on the person's posting style, and are limited by character-count (the current limit is 280 characters, but was originally 140 characters). Thus, individual characters, such as hashtags, emphasis on certain letters or symbols, etc. can have a significant impact on the semantics of the tweet, making character-based models suitable for tweet-related NLP tasks. In addition to this, if the majority of the tweet consists of words with extra characters or symbols, then using a word-based model would not work effectively as most of the words in the sentence would not have a corresponding embedding
For example, the following tweet from our "non-depressed" class was incorrectly classified with the word-based model, but was correctly classified with the character-based model. This was likely due to the extraneous characters present.

- `ayoooo any1 tryna get litty???`

The main disadvantage of using character-based embeddings is that training time increases significantly, since the sequence length inputted into the model is enlarged.

5.2.3 Analysis of RNN-based Models vs. CNN Model

From Figure 3, we can analyze the performance of the word-based CNN model (d). We see that the CNN's performance is comparable to the performance of the best RNN-based models (word-based GRU and character-based GRU). Even though CNNs are traditionally used for image recognition

tasks and do not store temporal information about the text, the CNN is still able to achieve good accuracy on our dataset because the convolutions performed by varying convolutional filters allows the model to learn efficient and useful representations of the text. While this CNN model is able to capture n-grams up to 5, other models are often unable to capture n-grams greater than 3 without significantly decreasing computational efficiency. These convolutions allow the CNN to process the text a lot faster, while word-based RNN-based models must read in the text word by word.

Another major difference between the CNN model and the RNN-based models is that the CNN learned word embeddings from scratch. Learned embeddings give the CNN an advantage, since it allows the model to learn the meaning of the words in the context of the dataset. One common phrase in the dataset was "depression nap", referring to nap to evade unwanted emotions or other symptoms related to depression. The learned word embedding of "nap" placed more emphasis on the meaning of "nap" in a depression context, while pre-trained embeddings would focus on the meaning of "nap" in a more general context. The following tweets from our "depressed" class was incorrectly classified with the word-based GRU, but was correctly classified with the CNN.

- `that was a fun depression nap but maybe i should eat that meal i was supposed to have four hours ago.`
- `after mixing those hard liquors your g is ready for another depression nap`

6 Conclusion and Further Improvements

6.1 Main Takeaways

The performances of all our models were relatively high, most likely due to the smaller size of our dataset. The best models for predicting depression in tweets were the word-based GRU model, with 98 % accuracy, and the word-based CNN model, with 97 % accuracy.

Through our experiments, we were able to make several comparisons between the different models we implemented. GRU cells performed better than RNN cells due to their ability to capture more long-term dependencies. For GRUs, using character embeddings resulted in more stability in the model's overall performance, but using word embeddings resulted in higher final accuracy. Finally, the CNN model produced comparable results as the high-performing GRU models due to the efficiency of convolutional filters in capturing sentence-level features, and learned word embeddings.

6.2 Model Improvements

Our model could use some improvements in future iterations. One worthwhile avenue to explore would be to use a CNN-LSTM Encoding-Decoding architecture as we had mentioned in a previous section because that had proven successful while performing sentiment classification. In addition to this, we could also experiment with a decaying learning rate to improve the stability of some of our loss and accuracy graphs (in addition to training for a longer time).

6.3 Dataset Improvements

There were several issues with the dataset that, if improved, would have made this project more practically applicable. The main issue with the dataset was that we were gathering data from tweets and prescribing a label of "depressed" or "not depressed" based on some relatively loose assumptions of the Twitter account. While these assumptions were based in logic, it would have been nice to have a dataset that was hand labeled by people with domain knowledge (ideally, have actual tweets from those who have been confirmed and known to have depression). If this sort of dataset existed, not only would we be more confident in our predictions actually being usable in the real world, but it would have allowed us to uncover more trends (for example, depressed people are not always "sad" in the traditional sense). Our models, while good at classifying "depression" as a type of sadness, would have benefited had a dataset like this existed.

References

[1] Britz D. (2015) Implementing a CNN for Text Classification in Tensorflow. WILDML. http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/

[2] Kim Yoon (2014) Convolutional Neural Networks for Sentence Classification. https://arxiv.org/pdf/1408.5882.pdf

[3] Minimax Char embeddings GitHub. https://github.com/minimaxir/char-embeddings

[4] Santos C.N. & Gatti M. (2014) Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, pages 6978, Dublin, Ireland, August 23-29 2014. http://www.aclweb.org/anthology/C14-1008.

[5] Twitterscraper library. https://github.com/taspinar/twitterscraper

[6] Vosoughi S. & Vijayaraghavan P. & Roy D. (2016) Tweet2Vec: Learning Tweet Embeddings Using Character-level CNN-LSTM Encoder-Decoder. ACM. https://arxiv.org/pdf/1607.07514.pdf.