Detecting Depression Through Tweets

Thomas Jacob
Department of Electrical Engineering
Indian Institute of Technology, Bombay
Mumbai, India
Email: thomasjac@ee.iitb.ac.in

Ishan Kapnadak
Department of Electrical Engineering
Indian Institute of Technology, Bombay
Mumbai, India
Email: ishankapnadak@iitb.ac.in

***Abstract*—This paper aims to classify Twitter users into different risk zones for depression based on their recent Tweets. We first build our own dataset by scraping tweets directly from Twitter from various pages related to depression and also through the help of keyword searches. We then constructed various deep learning models (LSTM, CNN, LSTM-CNN, GRU, BiLSTM and Attention-based Hybrid) and trained and evaluated them on our dataset. We examined the effect of using word embeddings vs. character embeddings vs. subword-level embeddings on the performance and stability of each model. The best-performing models were the subword-based CNN with an accuracy of 99.46%, the subword-based LSTM-CNN with an accuracy of 99.46% and the subword-based GRU with an accuracy of 99.25%.**

I. Introduction

A. Motivation

We have all felt sadness at some point in our lives. Sadness is inevitable, while dealing with the loss of loved ones, or trying to go through life's tough challenges. It is often said that life can neither be all happy nor all sad. It is a bit of both. The occasional sadness we experience is in fact part of this cycle itself. However, the problem arises when this sadness seems never-ending. Long periods of guilt, hopelessness could in fact be more than just your regular sadness, and could actually be indicative of clinical depression, a treatable medical condition.

The World Health Organisation (W.H.O) in [1], states that, "Depression is a common illness worldwide, with more than 264 million people affected. Depression is different from usual mood fluctuations and short-lived emotional responses to challenges in everyday life. Especially when long-lasting and with moderate or severe intensity, depression may become a serious health condition. It can cause the affected person to suffer greatly and function poorly at work, at school and in the family. At its worst, depression can lead to suicide. Close to 800 000 people die due to suicide every year. Suicide is the second leading cause of death in 15-29-year-olds."

Although there are known and effective treatments available for depression, a significant portion of the population never receives such treatment. A few reasons for this are - lack of resources, lack of trained health-care providers and social stigma associated with mental health. However, another often overlooked barrier to the treatment of depression is the lack of assessment. A significant number of depression cases are never identified and often go undiagnosed. Our project is aimed at getting more and more cases of depression identified and diagnosed.

B. Objective

Social media today is a minefield of data and a window into people's everyday lives. If used in the right way, social media provides an opportunity to flag potentially thousands of early depression cases. We have often scrolled through Twitter and saw a tweet by a person saying that they are "sick of life". In our busy day-to-day lives, we wouldn't pay much heed to such tweets. But these tweets are more often than not, a subtle call for help. We would like to identify and flag such tweets to keep a track of any early signs of depression in anyone. Note that our objective is not to diagnose depression but to only *flag* cases of early onset or potential cases of depression so that they can receive appropriate help.

The important question at this point was - what makes a tweet indicative of depression? A variety of research papers exist which look at lexical markers of depression. People suffering from depression often subconsciously use linguistic patterns indicative of depression. It is possible to build a dictionary containing all these markers, however this proved to be too extensive. An important point to note is that depression is something that is most frequently self-reported and self-assessed (at least in its initial stages). Hence, if a person often says that they are depressed, it is quite likely that they really are depressed. Additionally, words like "helpless", "lonely", "suicidal" are consistently linked with people who are suffering from feelings associated with depression.

C. Overview of Analysis

Keeping these points in mind, we decided to use some of these keywords and phrases to build our dataset. Tweets that were indicative of depression were scraped directly from Twitter using TWINT ([2]), an advanced Twitter scraping tool available online. We also used a Kaggle dataset ([3]) to obtain some random tweets that were not indicative of depression. Through this project, we looked at the performance of various neural network architectures on classifying tweets as indicative of depression. Further, we also looked at the effect of using word embeddings, character embeddings and subword-level embeddings on the performance of our models. We used accuracy as our primary metric for evaluating and comparing the performance of different architectures. However, since the depressive class of tweets was rather sparse, we also looked at other metrics such as precision, recall and F1 score to help us compare and evaluate different architectures. Additionally, the training and validation accuracy and loss of each architecture was monitored over each epoch. Using this, we also evaluated the architectures based on stability during training. We finally deployed our models and used them in a simple categorization algorithm to see how well our models perform on real Twitter users.

D. Related Works

The task of detecting depression through tweets falls under the broad spectrum of sentiment analysis, an area which has been explored a lot. With an increasing onset of cases of clinical depression in an age of virtual devices, and many people feeling safer in disclosing their thoughts to strangers on the internet through tweets, this particular task has garnered a lot of attention.

One paper, [4], employed an approach of treating each user individually in detail, extracting about 3000 of their most recent tweets to predict if they were suffering from depression or not. They used multiple machine learning approaches, namely Decision Trees, Naive Bayes and Support Vector Machines, and treated it as a binary classification problem. The best results were obtained using the SVM-L classifier, amounting to an accuracy of 82% and an F1 score of 0.79.

A number of other papers take a different approach wherein, instead of looking at each user in detail, a large corpus of tweets indicating depression are extracted along with some random tweets by scraping followed by the use of word embeddings to represent the data. Reference [5] compares different deep learning models (RNN, CNN and GRU) to generate predictions. They further compared word-based embeddings with character-based embeddings as well as pre-trained embeddings with learned embeddings. The best results were obtained with the word-based GRU model (98% accuracy) and the word-based CNN model (97% accuracy).

II. Datasets

We used two sources to acquire our data. Tweets indicative of depression were scraped from Twitter itself with the help of keyword-searches. We decided to use a total of eight keywords and phrases to help us build our dataset - these were : ‘depressed’, ‘anxious’, ‘lonely’, ‘helpless’, ‘suicidal’, ‘kill myself’, ‘sick of life’, and ‘life sucks’. Using [2], we scraped a total of $11,200$ tweets from the years 2018 and 2020 that contained one of these eight keywords and phrases. However, an extensive effort was required to clean this dataset manually. A significant number of the tweets which were scraped were irrelevant to our task at hand. For example, a large chunk of tweets that contained these keywords were in fact depression awareness posts. Secondly, some of these tweets were made in humour and some of them also involved movie references. Such tweets were manually discarded from the dataset. We also scraped another 500 tweets from 5 Twitter pages that were specifically about depression. For example, these pages had usernames such as “depressingmsgs”, “cuttingquote”, “suicidalconcept”. The motivation behind also including tweets from these pages was to increase the diversity of our data. We wanted our data to have more positive examples than just tweets which contained one of those eight keywords or phrases. Pages like these proved to be the perfect source for such data. Finally, we were left with a total of $1,926$ tweets that we believed to be truly indicative of depression. An example of such a tweet was

- `I've been so depressed lately but I try my best not to show it.`

Additionally, we gathered $8,040$ random tweets from [3]. These tweets were considered as ‘not indicative of depression’. An example of a tweet from this dataset was

- `I really really like the song Love Story by Taylor Swift`

Both these datasets contained a vast number of attributes such as - date, time, id, tweet, username, user_id, conversation_id, hashtags, cashtags, timezone, likes_count. However, only the ‘tweet’ attribute, which contained the text of the tweet, was important to us. Hence, we merged the scraped data and Kaggle data and only retained the ‘tweet’ attribute.

Next, we sanitised each tweet so that it can act as a suitable input to our model. The first step in the pre-processing chain involved some basic cleaning up of the tweet. For this task, we utilised tweet-preprocessor ([6]), a Python library for preprocessing of tweets. This preprocessor handled some simple tasks such as conversion to lower case, removal of URLs, Hashtags, Mentions, Emojis, Smileys and Numbers from the tweet. Next, we removed *stopwords* from the tweets. Since our main task was to identify depression in tweets and since stopwords such as ‘this’, ‘is’, ‘will’ did not add any meaning to the tweets on their own, it seemed reasonable to remove them from the tweet entirely. The next step involved expanding all contractions present in the tweet. A list of common contractions was easily available online. The final step in the pre-processing chain involved stemming. The primary goal of stemming was to reduce inflectional forms of a word to a common base (the root). For instance, ‘trouble’, ‘troubling’, ‘troubled’ are all reduced to ‘troubl’. Notice that the common base need not be a word in its own right. Let us demonstrate the effect of the entire pre-processing chain on a sample tweet from the random tweets dataset. The original tweet :

- `I'd have responded, if I were going`

After passing this tweet through the entire chain, the pre-processed tweet obtained was :

- `i would responded i go`

Observe how the entire tweet was converted to lower case. The contraction “I'd” was expanded to “I would” and the word “going” was converted to its common base - “go”. After the pre-processing of tweets was done, we were ready to train neural network architectures on these tweets.

III. Analysis Pipeline

A. *Setting Up The Data*

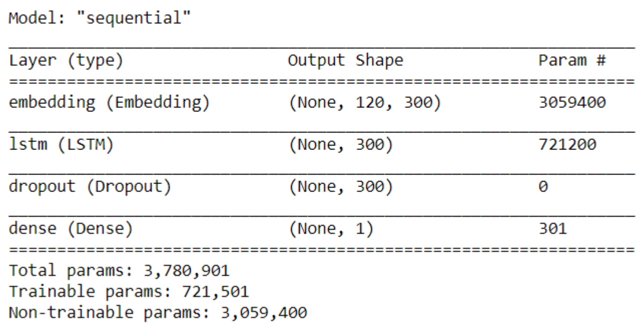
The entire data was split into three sets - training data (70%), validation data (15%) and test data (15%). The depressive tweets were assigned label 1 while the random tweets were assigned label 0. First, we experimented with using word embeddings. We used a tokenizer from the gensim library to tokenize the tweets. The maximum number of words in the vocabulary was set to $20,000$. The tokenizer was fit on all the training tweets, generating a vocabulary of $10,197$ unique tokens. Further, the Out Of Vocabulary (OOV) Token of the tokenizer was set to True. Thus, any words that were not part of the training vocabulary were assigned a common token of 1. Once the tokenizer was fit on training tweets, it was saved so that we could use the same tokenizer again later. All tweets were then converted to sequences using the tokenizer's texts_to_sequences method and were padded to a maximum length of 120 words. Next, we created an embedding matrix using the publicly available GoogleNews pre-trained word embeddings. Each word was encoded as a 300-dimensional vector. Thus, for our vocabulary of $10,197$ tokens, an embedding matrix of size $(10197, 300)$ was created.

B. *Baseline Model*

Before exploring neural network architectures, we first decided on a simple Logistic Regression model as our baseline. We used scikit-learn's logistic regression model with default hyperparameters (Regularisation $C = 0.1$ and Solver = 'lbfgs').

C. *LSTM Model*

Once our baseline was established, we moved on to exploring various neural network architectures. The first architecture we implemented was a standard LSTM (long short-term memory) model. An LSTM is a Recurrent Neural Network (RNN) architecture that is becoming increasingly popular to handle Natural Language Processing (NLP) tasks. The detailed architecture of our LSTM model is shown in Fig 1.



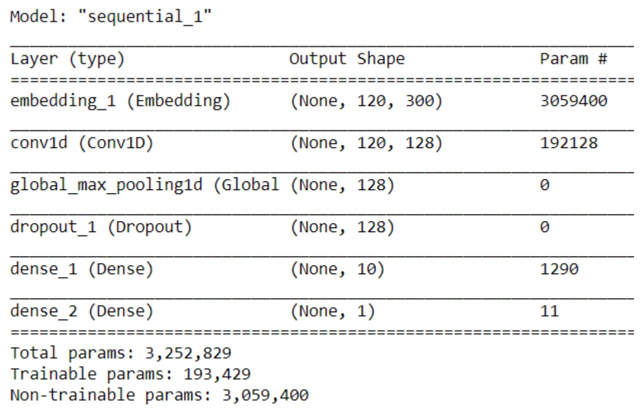
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 120, 300)          3059400
_________________________________________________________________
lstm (LSTM)                  (None, 300)               721200
_________________________________________________________________
dropout (Dropout)            (None, 300)               0
_________________________________________________________________
dense (Dense)                (None, 1)                 301
=================================================================
Total params: 3,780,901
Trainable params: 721,501
Non-trainable params: 3,059,400
```

Fig. 1. Architecture of the LSTM Model

The training data was first passed through an embedding layer with weights specified by the embedding matrix defined previously. The output of the embedding layer was then passed through an LSTM layer with a hidden state size of 300. The LSTM layer was followed by a Dropout layer with dropout probability 0.6. Finally, the output of the dropout layer was passed through a dense layer with a single output. A sigmoid activation was used at the end of the dense layer to squish the output between 0 and 1. This final output now represented the probability of the given tweet being indicative of depression. The model was trained for a total of 30 epochs with a batch-size of 32. The training and validation loss and accuracy were monitored throughout the training process. We also used an Early-Stopping mechanism with a patience of 2 to prevent overfitting. Once the training procedure was complete, the model was tested on the withheld 15% test data. A detailed classification report of the model along with its accuracy was then printed out.

D. *CNN Model*

Inspired by [5] and [7], the second architecture we looked at was a Convolutional Neural Network (CNN) Architecture. Although CNNs have been commonly used in computer vision tasks, their ability to recognise high-level spatial features and patterns has made them popular and surprisingly efficient in NLP tasks as well. The detailed architecture of the CNN model used is shown in Fig 2.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 120, 300)          3059400
_________________________________________________________________
conv1d (Conv1D)              (None, 120, 128)          192128
_________________________________________________________________
global_max_pooling1d (Global (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 11
=================================================================
Total params: 3,252,829
Trainable params: 193,429
Non-trainable params: 3,059,400
```

Fig. 2. Architecture of the CNN Model

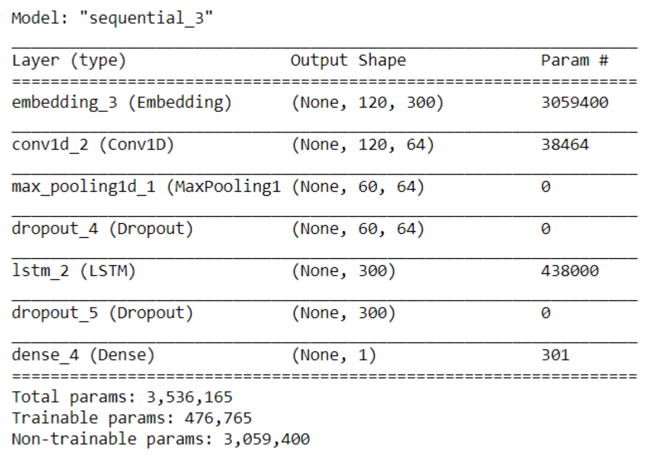
Aside from the architecture, everything else, such as the training and evaluation of the model remained exactly the same as the LSTM model. The architecture of the model largely follows the architecture specified in [7]. However, we used an additional Dense layer with 10 output units and a ReLU activation.

E. *LSTM+CNN Model*

Another increasingly popular architecture stacks an LSTM on top of a CNN architecture. Reference [8] claims that this hybrid LSTM-CNN model can effectively improve the accuracy in text classification tasks. Moreover, a similar LSTM-CNN hybrid architecture was also used in [9]. The LSTM-CNN architecture first uses the same embedding layer that was used in the other models. We then used a Convolutional Layer followed by a Maxpooling Layer and a Dropout layer.

This was followed by an LSTM layer and another Dropout layer. Finally, we use a Dense layer with a single output unit and sigmoid activation.

The detailed architecture of the hybrid LSTM-CNN model is shown in Fig 3.

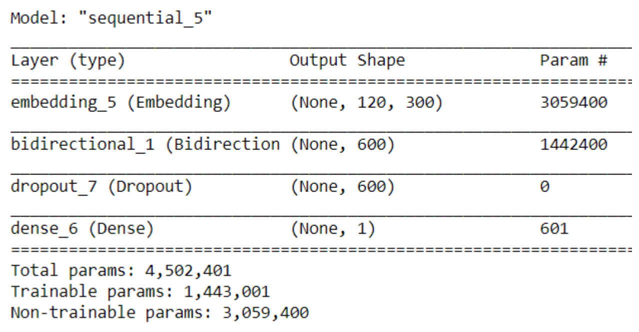


```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_3 (Embedding)      (None, 120, 300)          3059400   
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 120, 64)           38464     
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 60, 64)            0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 60, 64)            0         
_________________________________________________________________
lstm_2 (LSTM)                (None, 300)               438000    
_________________________________________________________________
dropout_5 (Dropout)          (None, 300)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 301       
=================================================================
Total params: 3,536,165
Trainable params: 476,765
Non-trainable params: 3,059,400
```

Fig. 3. Architecture of the hybrid LSTM-CNN Model

F. BiLSTM Model

Another variant of the LSTM Model we tried was the Bidirectional LSTM (BiLSTM) model. While a regular LSTM architecture holds memory only of the past inputs, a BiLSTM architecture has the added ability to process both past and future inputs at any time step. This feature seemed to be quite useful for sentence classification. In some cases, the context of a sentence is actually determined by words towards the end of the sentence. In such cases, a unidirectional forward LSTM would not perform well as it does not have access to the future words of the sentence. However, since a BiLSTM has access to the entire sentence at once, it would perform better than its unidirectional counterpart. Motivated by this, we defined the following architecture for the BiLSTM model, as shown in Fig 4.

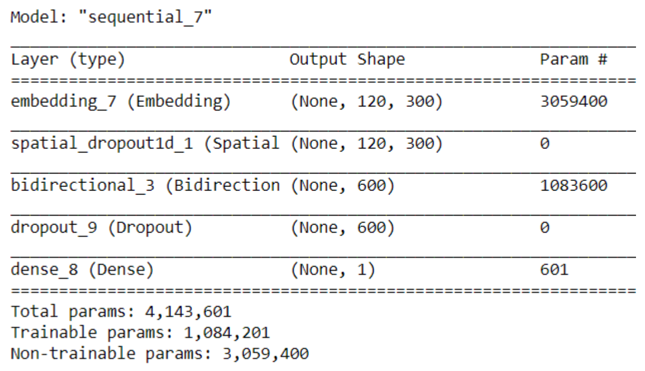


```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_5 (Embedding)      (None, 120, 300)          3059400   
_________________________________________________________________
bidirectional_1 (Bidirection (None, 600)               1442400   
_________________________________________________________________
dropout_7 (Dropout)          (None, 600)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 601       
=================================================================
Total params: 4,502,401
Trainable params: 1,443,001
Non-trainable params: 3,059,400
```

Fig. 4. Architecture of the BiLSTM Model

G. GRU Model

We also explored a GRU (Gated Recurrent Unit). As mentioned in [5], GRUs prove to be more computationally efficient than their LSTM variants as they have only two gates (update and reset) as compared to the three gates (input, output and forget) of an LSTM. Moreover, the performance of both these units is nearly at par. The detailed architecture of the GRU model is shown in Fig 5.



```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_7 (Embedding)      (None, 120, 300)          3059400   
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 120, 300)          0         
_________________________________________________________________
bidirectional_3 (Bidirection (None, 600)               1083600   
_________________________________________________________________
dropout_9 (Dropout)          (None, 600)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 601       
=================================================================
Total params: 4,143,601
Trainable params: 1,084,201
Non-trainable params: 3,059,400
```

Fig. 5. Architecture of the GRU Model

Note that we have used a Bidirectional architecture for the GRU model. Thus, we will compare this architecture to that of the BiLSTM model specified in section III-F.

H. BiLSTM Model with Attention

Many traditional LSTM architectures are able to recognise sequential patterns in text. However, there is a trade-off. While using an LSTM architecture, we lose out on the importance of each word to the sentiment of the sentence. Some words will have a higher impact in determining the sentiment of a sentence than others. Ideally, we would want to give a higher weight to these words. This task is fulfilled by adding an *attention* mechanism. Yang et al. (2016) state that "not all words contribute equally to the representation of the sentence meaning. Hence, we introduce attention mechanism to extract such words that are important to the meaning of the sentence and aggregate the representation of those informative words to form a sentence vector." [10, p. 3]. An attention-based hybrid BiLSTM+CNN model was also used in [11]. Motivated by this, we defined the architecture for our attention-based model as shown in Fig 6.

I. Character and Subword Embeddingss

Besides using word embeddings, we also explored character and subword embeddings to represent sentences accurately. The primary motivation for doing so was that a sizeable number of tweets are informal and contain words that might be out of vocabulary (OOV) for pre-trained word embeddings. This issue is resolved by character embeddings which only embed each character in the sentence rather than the words themselves. However, character embeddings come with the downside of being less capable of capturing a word's sentiment information as compared to word embeddings. Subword-level embeddings somewhat enjoy the best of both worlds with the ability to capture the semantics of a sentence to an extent,

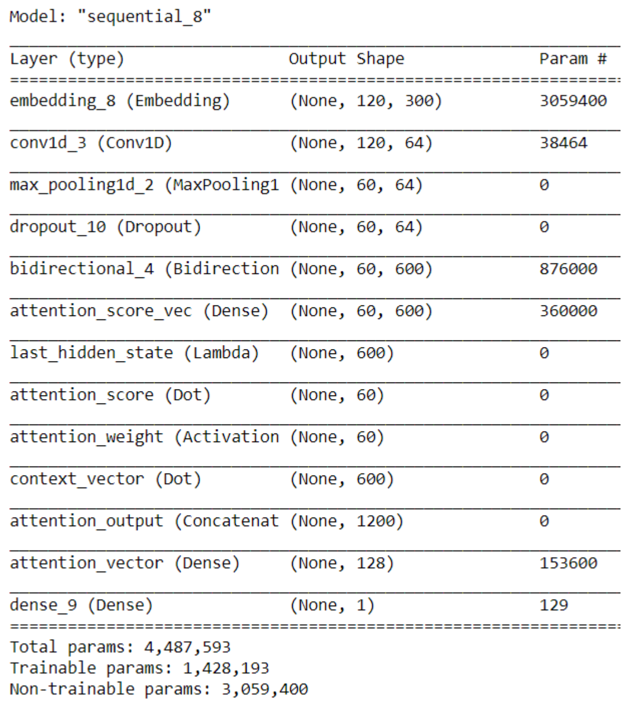

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_8 (Embedding)      (None, 120, 300)          3059400
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 120, 64)           38464
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 60, 64)            0
_________________________________________________________________
dropout_10 (Dropout)         (None, 60, 64)            0
_________________________________________________________________
bidirectional_4 (Bidirection (None, 60, 600)           876000
_________________________________________________________________
attention_score_vec (Dense)  (None, 60, 600)           360000
_________________________________________________________________
last_hidden_state (Lambda)   (None, 600)               0
_________________________________________________________________
attention_score (Dot)        (None, 60)                0
_________________________________________________________________
attention_weight (Activation (None, 60)                0
_________________________________________________________________
context_vector (Dot)         (None, 600)               0
_________________________________________________________________
attention_output (Concatenat (None, 1200)              0
_________________________________________________________________
attention_vector (Dense)     (None, 128)               153600
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 129
=================================================================
Total params: 4,487,593
Trainable params: 1,428,193
Non-trainable params: 3,059,400
```

Fig. 6. Architecture of the Attention-based BiLSTM+CNN Model

along with being able to represent OOV words. This requirement is crucial to our objective, a sentiment classification problem in an informal setting, which makes it more likely for users to use OOV words. We used Byte Pair Encoding (BPE), a common Subword Embedding technique, making use of helper code obtained from [12]. Once our models were trained and evaluated with all three embeddings, we proceeded to make a comparative analysis between word, character, and subword-level embeddings.

Since we were embedding sentences based on characters and subwords only, we performed minimal pre-processing of the tweets and only removed URLs present in the tweet and converted them to lower case. Further, architectures for character and subword-level embeddings were largely the same as word embeddings, except that the outputs were one-hot encoded and thus, the final dense layer had two output units.

J. *User Classifier*

Once all models were trained and evaluated, we moved on to deploying these models to classify users based on their tweets. The idea was as follows - we looked at the 500 most recent tweets of any particular user. If the number of tweets scraped were less than 10, we decided to not classify that user owing to insufficient data. If we could scrape at least 10 tweets from a particular user, we proceeded to classify that user into one of four 'risk' zones. If less than 2% of the user's recent tweets were classified as indicative of depression (by our trained model), we categorized the user as having 'low to no' risk of depression. If between 2% to 10% of the user's recent tweets were classified as indicative of depression, we categorized them as having 'mild' risk of depression. If between 10% to 35% of the user's recent tweets were classified as indicative of depression, we categorized them as having 'moderate' risk of depression. Anything above 35% and the user was classified as having 'significant' risk of depression.

IV. Results

A. *Metrics Used*

We primarily used four metrics to evaluate and compare our models. These were - accuracy (η), precision (P), recall (R) and $F1$-Score. These metrics are defined as follows:

$$\eta = \frac{TP + TN}{TP + TN + FP + FN}$$

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \cdot P \cdot R}{P + R}$$

where TP, TN, FP, FN denote the number of true positives, true negatives, false positives and false negatives respectively. Furthermore, we also kept a tab of training and validation accuracy and loss while training the model. A graph of these parameters were plotted against the number of epochs. This helped in comparing models based on stability. Once all models were trained and evaluated with all three embeddings, we undertook a comparative study between the three embeddings.

B. *Word Embeddings*

The baseline logistic regression model achieved an accuracy 81.34% with word embeddings. Surprisingly, all other architectures we used had roughly similar performances and there were no pronounced differences between their accuracies. Moreover, all architectures performed significantly better than the baseline logistic regression model. The LSTM architecture achieved an accuracy of 94.38%. The CNN model performed slightly better than the LSTM, achieving an accuracy of 94.58%. The hybrid LSTM-CNN model achieved a slightly lower accuracy of 93.98%. The BiLSTM model did not perform as well as the LSTM and achieved an accuracy of 93.38% which was an entire 1% less than the 94.38% accuracy achieved by the unidirectional LSTM. This went *against* our intuition that the BiLSTM would perform better at text classification tasks. The (bidirectional) GRU model achieved an accuracy of 93.44%, which was better than the BiLSTM model. Thus, the GRU model outshone its LSTM counterpart despite having one lesser gate. Finally, the attention-based BiLSTM-CNN model achieved a lower accuracy of 92.64%. Thus, the CNN model came out on top as the best-performing architecture, closely followed by the LSTM and the hybrid LSTM-CNN models. This was a surprising result because the CNN architecture, which was initially meant for computer vision tasks, outperformed both LSTM and GRU architectures which are well suited to NLP tasks. This demonstrated the versatility of the CNN architecture.

C. Character Embeddings

The performance of the baseline logistic regression model increased on using character embeddings, with an increased accuracy of 84.35%. The performance of the LSTM model dropped sharply as it displayed an accuracy of just 87.66% which was only slightly better than the baseline accuracy of 84.35% and significantly lesser than the 94.38% it achieved with word embeddings. The performance of the CNN, GRU and attention-based models increased slightly, displaying accuracies of 97.09%, 96.49% and 94.18% respectively. The BiLSTM model achieved an accuracy of 89.57% which was just slightly above the baseline model while the hybrid LSTM-CNN model achieved an accuracy of 93.48% which was nearly the same as the 93.98% accuracy it achieved with word embeddings.

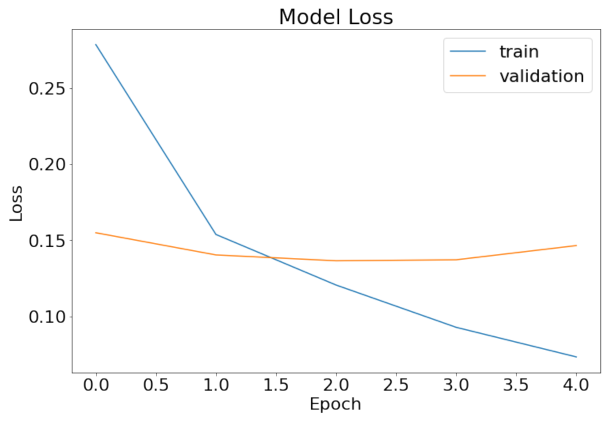
D. Subword-Level Embeddings

The performance of the baseline logistic regression model increased on using subword-level embeddings as well, achieving an accuracy of 87.47%. The performance of the LSTM model again dropped sharply, achieving an accuracy of just 87.58%, which provides hardly any significant improvement over the baseline model. Further, the learning rate for the LSTM model seemed to be too low and the loss and accuracy on both training and validation data seemed to remain more or less the same over numerous epochs. The performance of the BiLSTM model increased to 94.97% while the performance of the attention-based model dropped to 91.01%. The remaining three models - CNN, LSTM+CNN and GRU - displayed a sharp *increase* in performance, achieving astounding accuracies of 99.46%, 99.46% and 99.25% respectively. (See Appendix A for detailed classification reports of each model with each embedding).

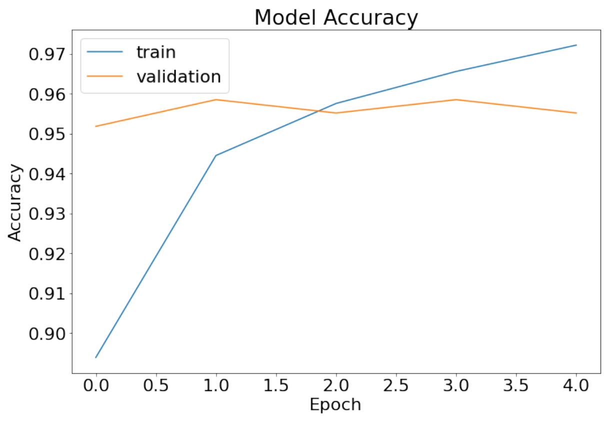
E. Qualitative Stability Analysis

We also monitored the behaviour of the models during their training phase with the help of graphs. An example of such a graph is shown in Fig 7, which shows the training and validation loss and accuracy of the word-embedded CNN model against the number of epochs.

We analysed these loss curves and drew conclusions about the stability of models. The CNN model was by far the most stable model amongst all architectures, displaying extremely smooth loss and accuracy curves. The LSTM and BiLSTM models were quite unstable, showing spikes and erratic jumps in their loss curves. Adding a CNN layer on top of an LSTM layer improved the stability of the model, as was seen in the LSTM-CNN and the attention-based hybrid model. The GRU model was more stable than its LSTM counterpart (BiLSTM model) but still showed spikes in its loss curves. These trends roughly remained the same across all three embeddings, indicating that the embedding had little to no effect on the stability of the model (See Appendix B for figures).



(a) Loss Curve



(b) Accuracy Curve

Fig. 7. Measurements of training and validation loss and accuracy for the CNN model with word embeddings.

F. User Classifications

Finally, we deployed our models to classify users into risk zones based on depression. We used a total of 10 Twitter handles for our experiment. First, we tested our classifier on 5 celebrities who have opened up about their depression previously in interviews. These were - Dwayne Johnson (@TheRock), Katy Perry (@katyperry), Lady Gaga (@ladygaga), JK Rowling (@jk_rowling) and Deepika Padukone (@deepikapadukone). We also took five random Twitter pages. These were - NDTV (@ndtv), Indian Premier League (@IPL), Bon Appétit (@bonappetit) (a food blog), Animal Planet (@AnimalPlanet) and Fun Facts (@Funfacts).

We first tested our CNN model with word embeddings. It classified Katy Perry, Lady Gaga and JK Rowling as having a mild risk of depression. However, it classified Dwayne Johnson and Deepika Padukone as having low to no risk of depression. The model also correctly classified all five random pages as having low to no risk of depression.

Next, we tested our CNN model with subword embeddings as well as our GRU model with subword embeddings. Both the CNN and the GRU models classified Dwayne Johnson and Katy Perry as having a moderate risk of depression. The CNN model classified Lady Gaga and JK Rowling as having a mild risk of depression and Deepika Padukone as having a moderate risk of depression. The GRU model classified Lady Gaga and JK Rowling as having a moderate risk of depression and classified Deepika Padukone as having a mild risk of depression. The CNN model correctly classified all five random pages as having low to no risk of depression. The GRU model classified NDTV, Bon Appétit and Fun Facts as having low to no risk of depression but classified IPL and Animal Planet as having mild risk of depression.

V. Discussion

A. Key Takeaways

Almost all of our architectures displayed quite a high performance. We suspect that this was due to the smaller size of our dataset and lesser diversity present in the dataset. The best models were the subword-based CNN model with an accuracy of 99.46%, the subword-based LSTM+CNN model with an accuracy of 99.46% and the subword-based GRU model with an accuracy of 99.25%.

We were also able to compare the strengths and weaknesses of various architectures through our experiments. For example, the GRU model performed better than the BiLSTM model despite having one gate lesser. The CNN model was the most stable model and also the highest performing one. This displayed the versatility of convolutional networks in identifying sentence-level features. The LSTM model was inherently unstable but adding a convolutional layer greatly improved its stability. The performance of the LSTM and BiLSTM models significantly deteriorated with character embeddings. Since LSTMs are equipped to model only short-range dependencies, we suspect that this drop in performance was due to the fact that embedding sentences as characters or subwords would prevent the LSTM from learning any dependencies between words at all since its neighbourhood now consists of really just a word or two at max. We hence conclude with a good amount of confidence that word embeddings were a better option for LSTM and BiLSTM models as compared to character embeddings. For the other models, the character embeddings did improve performance, but since the increase in performance was not too pronounced, this conclusion remained weak. Subword-level embeddings did not hamper the performance of any architecture significantly (apart from the LSTM) and they greatly increased the performance of the CNN, GRU and LSTM-CNN models. We could thus make a strong conclusion that subword level embeddings are the best-suited embedding for our task for nearly all architectures.

B. Strengths

Our subword-level embedded models were quite adept at identifying tweets indicative of depression. Moreover, our user classification algorithm, though simple, worked effectively well. It predicted a moderate or a mild risk of depression in all of the five celebrities we considered. Since celebrities have a well-curated social media presence, this result is not a strong conclusion. However, we believe that the mass public are a lot more open about their mental health issues on social media as compared to celebrities. Keeping this in mind, we expect our model to perform *better* on a more widespread population.

We believe that our approach of classifying users into risk zones based on their tweets is unique and has the potential to help many. Since users are classified into different risk zones, each category can receive help that is suited for them. Our approach has the potential to identify early signs of depression through something as simple as a tweet and we believe that with a little bit of effort, we can make sure that people suffering from depression, at any level of intensity, are identified and given the care that they deserve.

C. Improvements and Further Experimentation

Our project could use some improvements in several key areas. We elaborate on three of these areas - dataset improvements, model improvements and classification algorithm improvements.

Currently, we have only scraped tweets corresponding to eight keywords and key-phrases and five depression-related pages. This was thus, a relatively small dataset and not diverse enough. To make our models more robust and versatile, an expansion in data is necessary. A much more extensive dictionary of lexical markers indicative of depression can be created. Moreover, these tweets can be labelled as indicative of depression by experts and medical practitioners who would be able to understand fine nuances in each tweet and classify them appropriately with the help of their vast experience and knowledge. Instead of depression-related pages, we could also train our model on twitter users who have already been diagnosed with depression.

Besides our dataset, we could also make some improvements to our models in future iterations. A Multiplicative LSTM is a worthwhile model to explore. We may also explore the effect of pre-trained embeddings and learned embeddings, as done in [5]. We also wish to explore various other subword level embedding techniques besides Byte Pair Encoding (BPE), such as Sentencepiece. We also wish to incorporate a ‘time’ element into our model. That is, we would like to explore if the time of the day when a user tweets most frequently has any correlation with whether the user is depressed or not. We also wish to explore using Transformers for our task.

Finally, we can also make improvements in our user classification algorithm. Our current algorithm is very crude and simplistic. This algorithm can be made more intelligent by allowing the algorithm to learn the classification boundaries for different risk zones on its own. To achieve this, several Twitter users could first be categorised into the risk zones we outlined based on the severity of their cases. We can then program our model to classify each tweet into one of these risk zones, rather than just a simple binary classification. Finally, the most recent 500 tweets of a user will be scraped and classified and the user will be assigned the risk zone which has a majority of the user’s tweets assigned to it.

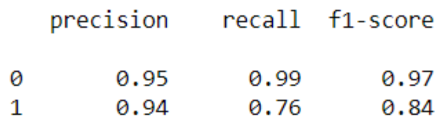
Acknowledgment

We would like to thank our instructors, Prof. Amit Sethi, Prof. Manjesh K Hanawal, Prof. S Sudarshan and Prof. Sunita Sarawagi of IIT Bombay. This project would not have been possible without their guidance. We would also like to thank our Teaching Assistant, Drumil Trivedi of IIT Bombay, for their constant assistance with our project. We would like to thank our classmates, Aaron Jerry Ninan, Tirthankar Adhikari, Moysha Gera, Hiya Gada and Harshda Saxena of IIT Bombay for helping us decide what keywords and phrases to use to build our dataset. Lastly, we would like to thank our parents and family for their constant support and motivation.

References

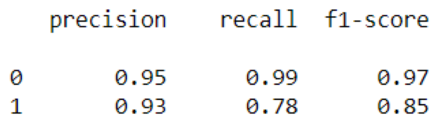
[1] World Health Organisation - Depression Fact Sheets. https://www.who.int/news-room/fact-sheets/detail/depression
[2] TWINT - Twitter Intelligence Tool. https://github.com/twintproject/twint
[3] Kaggle Data Set - Twitter Sentiment Extraction. https://www.kaggle.com/c/tweet-sentiment-extraction/data?select=train.csv
[4] Hatoon AlSagri and Mourad Ykhlef. "Machine Learning-based Approach for Depression Detection in Twitter Using Content and Activity Features," 2020.
[5] Diveesh Singh and Aileen Wang. "Detecting Depression Through Tweets," 2018.
[6] tweet-preprocessor. https://pypi.org/project/tweet-preprocessor/
[7] Kim Yoon. "Convolutional Neural Networks for Sentence Classification," 2014.
[8] J. Zhang, Y. Li, J. Tian and T. Li, "LSTM-CNN Hybrid Model for Text Classification," 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chongqing, 2018, pp. 1675-1680, doi: 10.1109/IAEAC.2018.8577620.
[9] Anne Bonner. "You Are What You Tweet: Detecting Depression in Social Media via Twitter Usage," 2019.
[10] Zichao Yang, Diyi Yang, Chris Dyer, Xiadong He, Alex Smola and Eduard Hovy. "Heirarchial Attention Networks for Document Classification," 2016.
[11] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, Sang-ug Kang and Jong Wook Kim. "Bi-LSTM Model to Increase Accuracy in Text Classification: Combining Word2Vec CNN with Attention Mechanism," 2020.
[12] Xu, Liang. "Keras Implementations for NLP Models, " 2018. https://github.com/BrambleXu/nlp-beginner-guide-keras
[13] Pei-Jo Yang (MIT). "Detect Depression In Twitter Posts," 2020. https://github.com/peijoy/DetectDepressionInTwitterPosts

Appendix A
Classification Reports



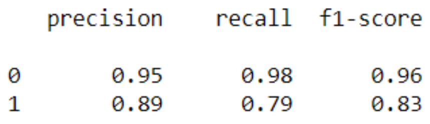
```
      precision    recall  f1-score

0        0.95       0.99      0.97
1        0.94       0.76      0.84
```

(a) LSTM Architecture



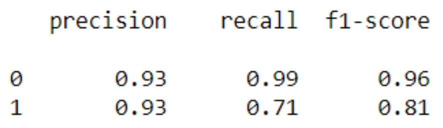
```
      precision    recall  f1-score

0        0.95       0.99      0.97
1        0.93       0.78      0.85
```

(b) CNN Architecture



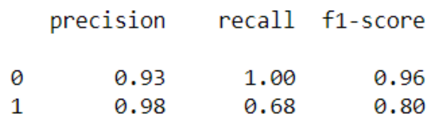
```
      precision    recall  f1-score

0        0.95       0.98      0.96
1        0.89       0.79      0.83
```

(c) LSTM+CNN Hybrid Architecture



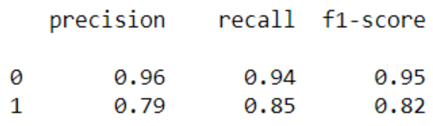
```
      precision    recall  f1-score

0        0.93       0.99      0.96
1        0.93       0.71      0.81
```

(d) BiLSTM Architecture



```
      precision    recall  f1-score

0        0.93       1.00      0.96
1        0.98       0.68      0.80
```

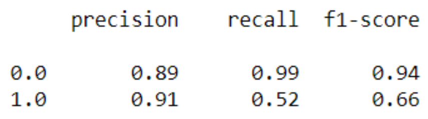
(e) GRU Architecture



```
      precision    recall  f1-score

0        0.96       0.94      0.95
1        0.79       0.85      0.82
```

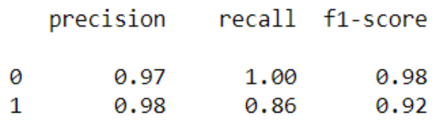
(f) Attention-based Hybrid Architecture

Fig. 8. Classification reports of the six architectures with word embeddings



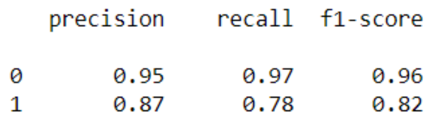
```
        precision    recall  f1-score

0.0        0.89       0.99      0.94
1.0        0.91       0.52      0.66
```

(a) LSTM Architecture



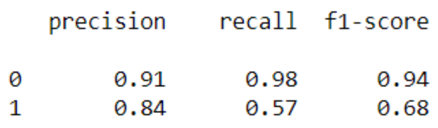
```
      precision    recall  f1-score

0        0.97       1.00      0.98
1        0.98       0.86      0.92
```

(b) CNN Architecture



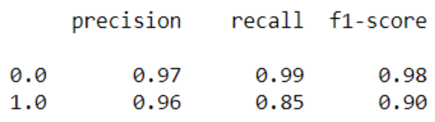
```
      precision    recall  f1-score

0        0.95       0.97      0.96
1        0.87       0.78      0.82
```

(c) LSTM+CNN Hybrid Architecture



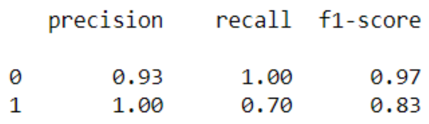
```
      precision    recall  f1-score

0        0.91       0.98      0.94
1        0.84       0.57      0.68
```

(d) BiLSTM Architecture



```
        precision    recall  f1-score

0.0        0.97       0.99      0.98
1.0        0.96       0.85      0.90
```

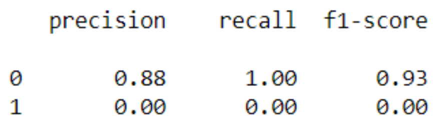
(e) GRU Architecture



```
      precision    recall  f1-score

0        0.93       1.00      0.97
1        1.00       0.70      0.83
```

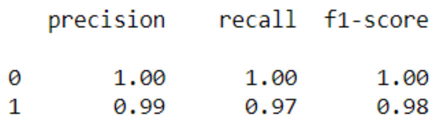
(f) Attention-based Hybrid Architecture

Fig. 9. Classification reports of the six architectures with character embeddings



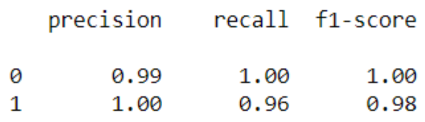
```
      precision    recall  f1-score

0        0.88       1.00      0.93
1        0.00       0.00      0.00
```

(a) LSTM Architecture



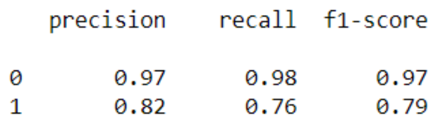
```
      precision    recall  f1-score

0        1.00       1.00      1.00
1        0.99       0.97      0.98
```

(b) CNN Architecture



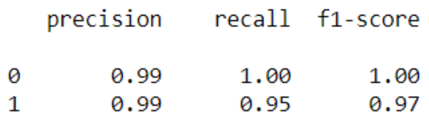
```
      precision    recall  f1-score

0        0.99       1.00      1.00
1        1.00       0.96      0.98
```

(c) LSTM+CNN Hybrid Architecture



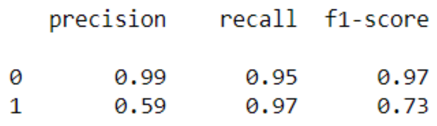
```
      precision    recall  f1-score

0        0.97       0.98      0.97
1        0.82       0.76      0.79
```

(d) BiLSTM Architecture



```
      precision    recall  f1-score

0        0.99       1.00      1.00
1        0.99       0.95      0.97
```

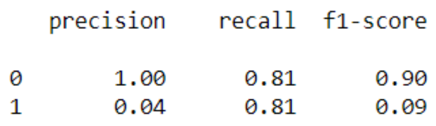
(e) GRU Architecture



```
      precision    recall  f1-score

0        0.99       0.95      0.97
1        0.59       0.97      0.73
```

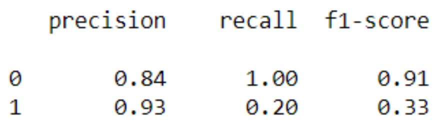
(f) Attention-based Hybrid Architecture

Fig. 10. Classification reports of the six architectures with subword-level embeddings



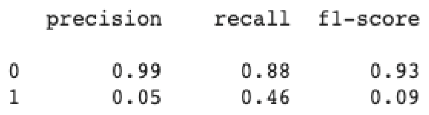
```
      precision    recall  f1-score

0        1.00       0.81      0.90
1        0.04       0.81      0.09
```

(a) Word Embeddings



```
      precision    recall  f1-score

0        0.84       1.00      0.91
1        0.93       0.20      0.33
```

(b) Character Embeddings



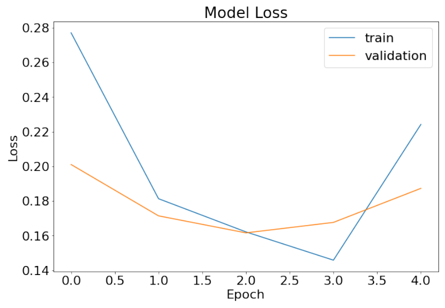
```
      precision    recall  f1-score

0        0.99       0.88      0.93
1        0.05       0.46      0.09
```

(c) Subword-Level Embeddings

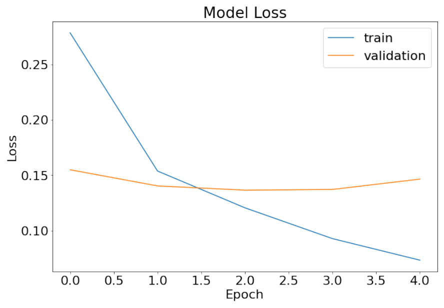
Fig. 11. Classification reports of the baseline logistic regression model for the three different embeddings

Appendix B
Performance Curves

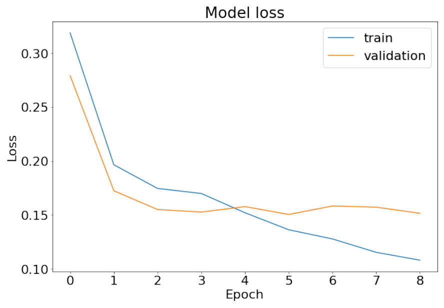
A. Word Embeddings



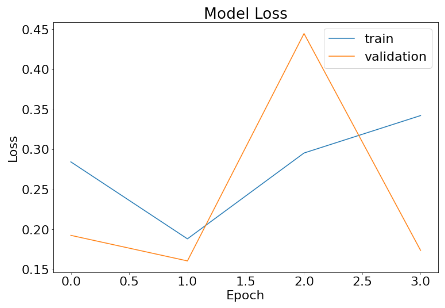
(a) LSTM Architecture



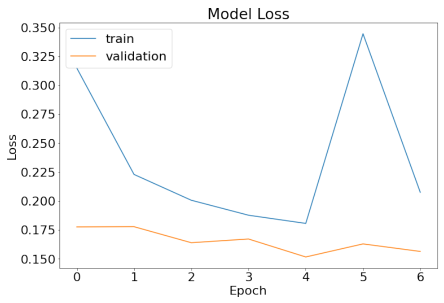
(b) CNN Architecture



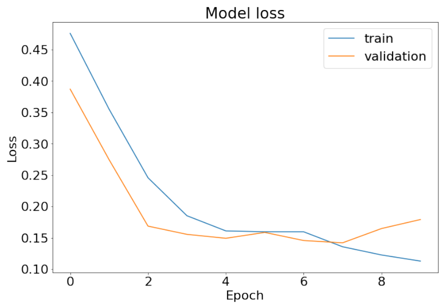
(c) LSTM+CNN Hybrid Architecture



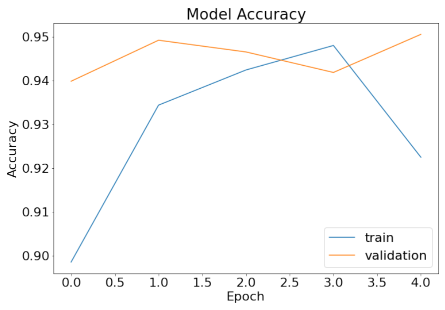
(d) BiLSTM Architecture



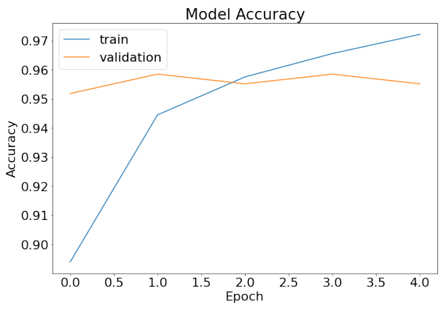
(e) GRU Architecture



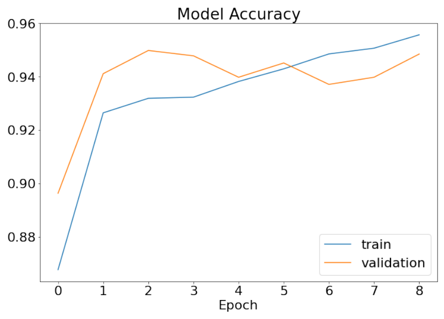
(f) Attention-based Hybrid Architecture



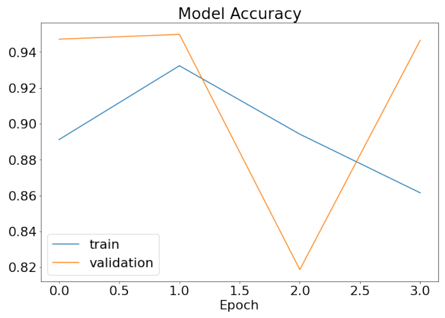
(g) LSTM Architecture



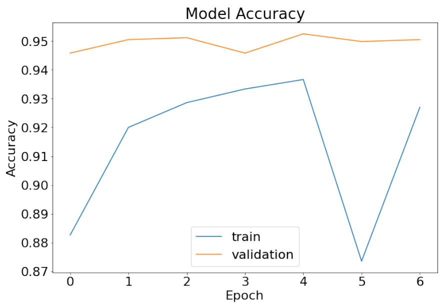
(h) CNN Architecture



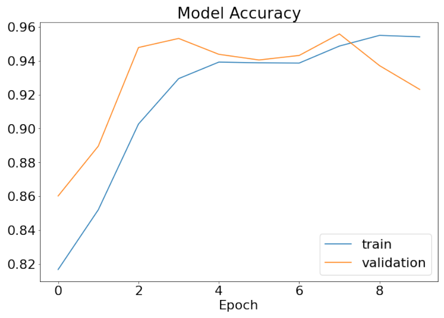
(i) LSTM+CNN Hybrid Architecture



(j) BiLSTM Architecture



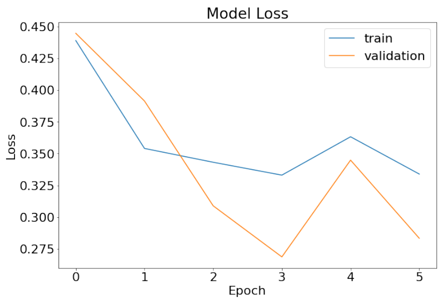
(k) GRU Architecture



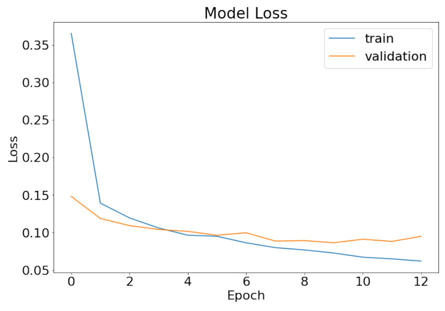
(l) Attention-based Hybrid Architecture

Fig. 12. Measurements of training and validation loss and accuracy for the six architectures used with word embeddings.

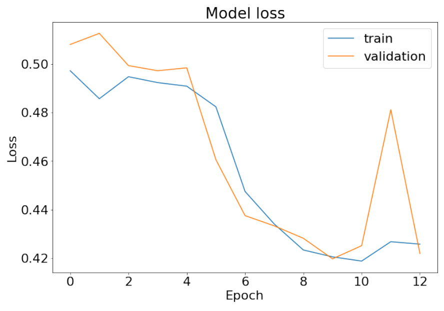
B. Character Embeddings



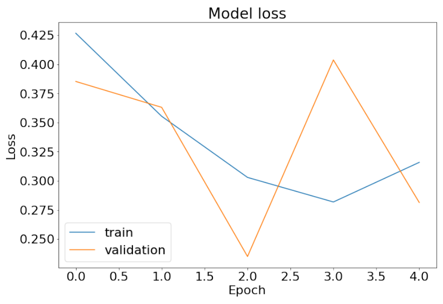
(a) LSTM Architecture



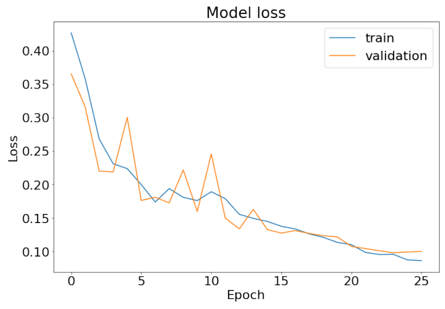
(b) CNN Architecture



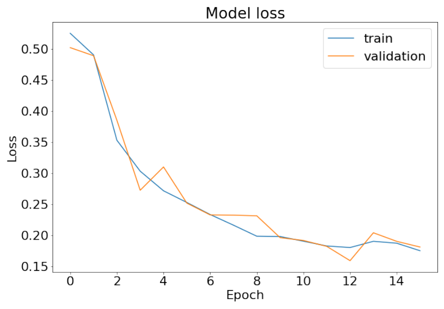
(c) LSTM+CNN Hybrid Architecture



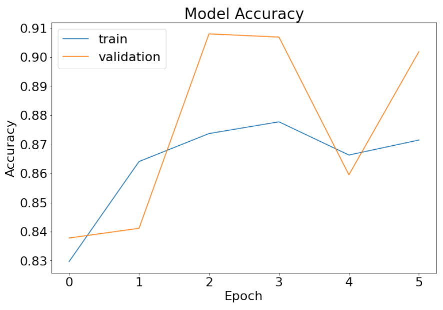
(d) BiLSTM Architecture



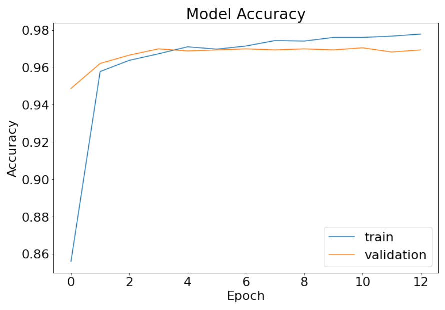
(e) GRU Architecture



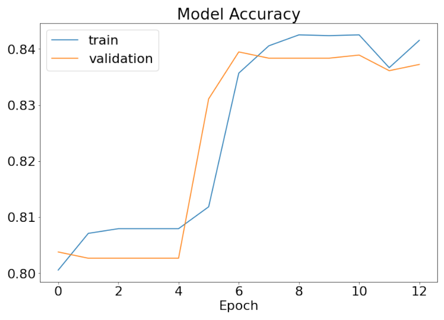
(f) Attention-based Hybrid Architecture



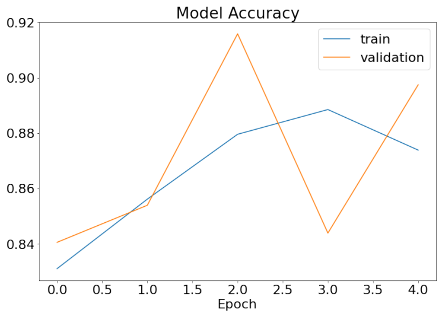
(g) LSTM Architecture



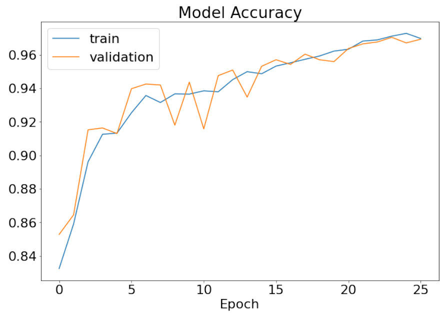
(h) CNN Architecture



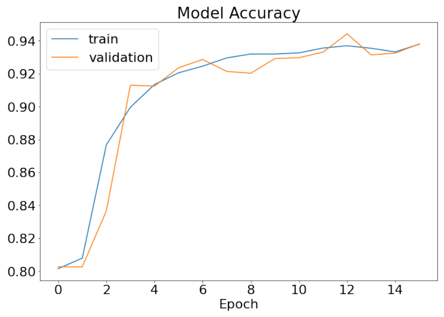
(i) LSTM+CNN Hybrid Architecture



(j) BiLSTM Architecture



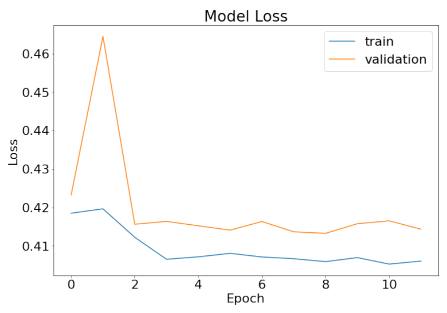
(k) GRU Architecture



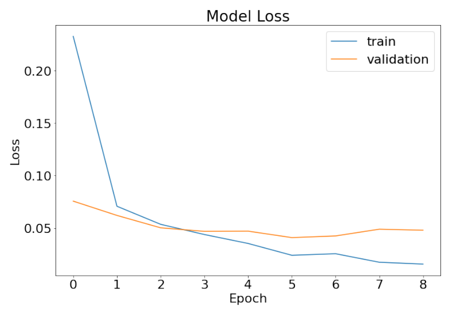
(l) Attention-based Hybrid Architecture

Fig. 13. Measurements of training and validation loss and accuracy for the six architectures used with character embeddings.

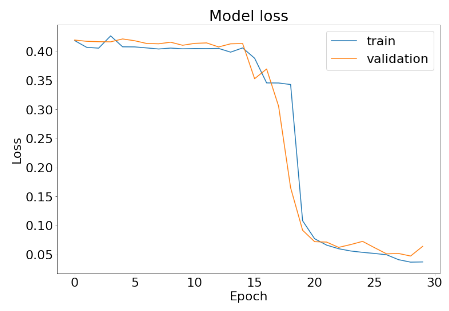
C. Subword-Level Embeddings



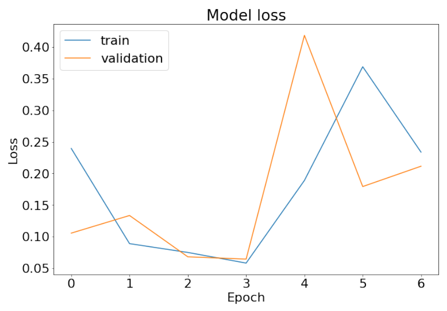
(a) LSTM Architecture



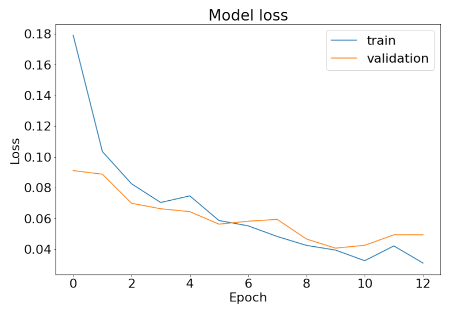
(b) CNN Architecture



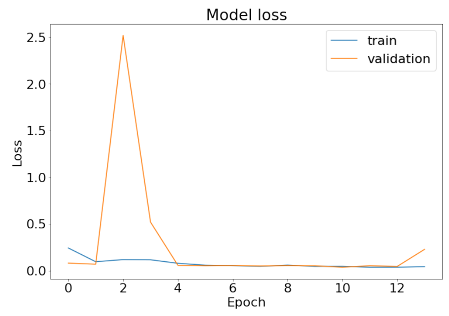
(c) LSTM+CNN Hybrid Architecture



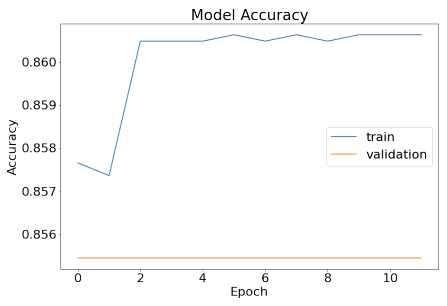
(d) BiLSTM Architecture



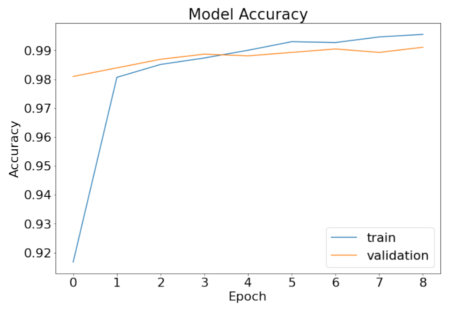
(e) GRU Architecture



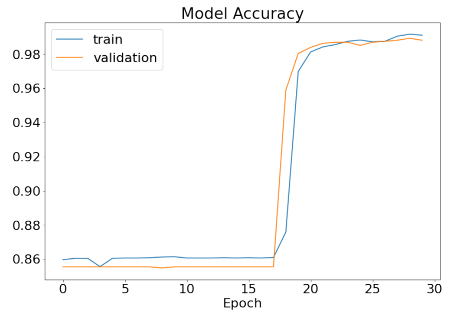
(f) Attention-based Hybrid Architecture



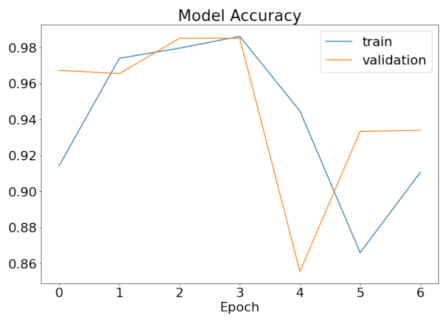
(g) LSTM Architecture



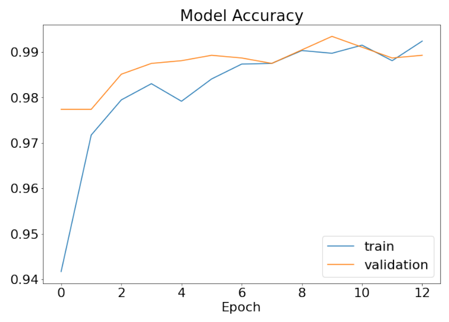
(h) CNN Architecture



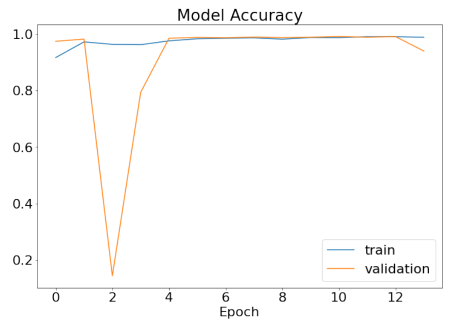
(i) LSTM+CNN Hybrid Architecture



(j) BiLSTM Architecture



(k) GRU Architecture



(l) Attention-based Hybrid Architecture

Fig. 14. Measurements of training and validation loss and accuracy for the six architectures used with subword-level embeddings.

Appendix C
List of Notebooks and Files

Following is a list of notebooks and files that were used in our project

1. `Scraping_Keywords.ipynb` - Jupyter Notebook for scraping of tweets using keyword searches
2. `Scraping_Pages.ipynb` - Jupyter Notebook for scraping of tweets from depression-related pages
3. `Preprocessing.ipynb` - Jupyter Notebook for preprocessing of tweets
4. `Word-Embeddings.ipynb` Jupyter Notebook to train and evaluate all models with word embeddings
5. `Character-Embeddings.ipynb` Jupyter Notebook to train and evaluate all models with character embeddings
6. `Subword-Embeddings.ipynb` Jupyter Notebook to train and evaluate all models with subword-level embeddings
7. `User-Classifier-Word.ipynb` Jupyter Notebook to deploy final model with word embeddings and classify users into risk zones
8. `User-Classifier-Subword.ipynb` Jupyter Notebook to deploy final model with subword-level embeddings and classify users into risk zones
9. `GoogleNews-vectors-negative300.bin.gz` Pretrained Word Embeddings from GoogleNews
10. `tokenizer.pickle` Saved tokenizer file
11. Helper files for Subword-level embedding.
12. CSV files containing data of all tweets
13. H5 files containing saved model files for all architectures and embeddings