Matthew Li
mhli@usc.edu

I chose to do a classification because a tweet should be either positive, negative, or neutral, and not in-between. In preprocessing, I used Glove, the same library from the RNN's notebook, to pre-process the data. I first loaded all of the relevant data and got rid of the rest of the data. Then, I tokenized them by turning the sentences into sequences of words and padding them to the size 50 to ensure that each array is the same size. Later, I also ensured that each of the embedded arrays I chose were also of size 50 to allow the model to train without errors. Finally, I loaded the word embeddings from glove and created embedded arrays for each of the tweets.

For post-processing, I chose 1/1000 of the dataset to allow the program to run faster. Furthermore, I created an array to store the valences, with [1,0,0] being equivalent to a valence of 0, [0,1,0] = 2, and [0,0,1] = 4. I used LSTMs in the model because it is important to see the context that words in a tweet are in order to determine whether they are positive or negative. For my model, I did:

```
model.add(LSTM(64, return_sequences = True, input_shape=(118, 50),
activation='relu'))
model.add(Dropout(.2))

model.add(LSTM(64, activation='relu'))
model.add(Dropout(.2))

model.add(Dense(3, activation = 'tanh'))
```

I added the dropouts to reduce overfitting and used relu to overcome the Vanishing Gradient Problem and to avoid the Long Term Dependency Problem. I used tanh on my Dense function in order to receive a final value of -1 to 1 to determine the positivity or negativity of a tweet. Like we did in the RNN notebook, I chose binary_crossentropy and RMSprop for the loss and optimizer. After messing around with the numbers, I ended with using 70% of the data on testing and 30% on training. I evaluated my model using the accuracy of both the training data and testing data, which ended up being around 70% for both.