Zaid Abdulrehman zabdulre@usc.edu

2) I chose to complete a classification task. I decided to do classification as it allowed me to most easily understand the model output. For example, in a regression task, the model may output a value which does not correspond exactly to one of the 3 expected valence values. As a result, it would become necessary to account for the bounds pertaining to each valence category. I believe this may have led to the need for ensemble methods or further parameter tuning, which may have led to added complexity. By sticking to classification, I took advantage of the simplicity of the classification process in order to focus on hyperparameter tuning and data preprocessing. I understood the problem to be a binary classification since none of the labels in the provided dataset had the value of ‘2’.

3) I had originally considered using the GloVe pre-trained word embedding. However, I decided against this as the model was 1GB, I wanted to ensure my project was easy to access and test. Also, I wanted to experiment with and understand the difference word embedding can have on the final model. I decided to create a vectorizer (Tokenizer). I initially settled on a vecotrizer which was simply trained on the vocabulary used in the testing sample. However, I realized that this approach led to a large variability in the model’s accuracy when training. So, I decided to use a vectorizer which is trained on the entire english dictionary. Furthermore, I incorporated an embedding layer which was trained in conjunction with the rest of the model (Not prepared separately). To preprocess the training and testing data, I: removed all stop words, removed all words not in the dictionary, and lemmatized all remaining words. I had also implemented a spell checker, however, this approach significantly increased training and testing time.

4) I trained the model solely on user tweets. The model takes an English sentence, corresponding to a tweet, and outputs a set of probabilities, pertaining to each valence category.

	Domain	Type
Tweet (Features)	String	String
Valence (Targets)	{0, 2, 4}	int

5) I used a neural network. The model is sequential. The first part of the model creates a tensor from the input data and feeds it to the vectorizer. The vectorizer simply creates a vector of integers where each integer represents the ‘id’ of the word. The second part of the model is responsible for creating the word embedding. This is a built-in layer of Keras which works as a very large look up table rather than a neural network. This layer is trainable. The input dimension of this model corresponds to the maximum output length of the vectorizer. For this layer, I adjusted the vocabulary size as well as the number of units. I found it very difficult to increase the size of the model while ensuring that the model could be trained within a reasonable time because the vocabulary size strongly correlated to the number of trainable parameters (I went over 24 million parameters at one point). The final part of the model is based on an LSTM layer and two dense layers. This final part takes the matrix output by the

embedding layer as an input and it outputs the final result. For the LSTM I configured the dropout and the number of units. For the next dense layer, I configured the number of units and set the activation function to sigmoid. For the final dense layer, I did not change the unit value from 2 and the activation function from softmax. I settled on this model structure based on my aim to keep the model easy to tune. I understood that an RNN would be required as the model requires ‘memory’ of previous characters/words in order to process sentences. Likewise, I first experimented with the GRU model, but I switched to the LSTM layer based on practical results. In order to produce the final output, a dense layer would be required. I did not choose to simply add one dense layer, however. My reasoning is that by only adding one dense layer between the large LSTM layer and the small output, I would constrain my model. So, I added an intermediary layer.

6) My loss function is categorical crossentropy and my optimizer is rmsprop.

7) I used stratified sampling to split the dataset into a stratified subset. I then split this subset further into a testing and training set.

8) I mainly used accuracy on the testing set to evaluate my model. My results are mixed. I noticed that the accuracy of my model was optimal when the model was trained between 5-13 epochs. When training my vectorizer on tweets rather than a dictionary, I noticed that variance in my training set affected my model’s accuracy by up to as much as 5%. Using this version of the vectorizer, I was able to achieve an accuracy of 75% on a testing set of 50,000 tweets. However, I switched to a vectorizer trained on a dictionary in favor of reciprocity and reliability. Currently, the model’s accuracy is at 69%. I would very much alter my approach on this project if I were to complete it again. Firstly, I would not incorporate the embedding layer as a part of the model. By doing so, I am severely limiting the feasible size of the model (Training time increases greatly due to the embedding). Overall, I believe that by tweaking my approach, I would address the current reliability, training time, and accuracy concerns.