Sentiment Analysis of Tweets - Jimmy Zhang

Introduction: The goal of my project is to classify the sentiment of a Tweet based on its textual content. This problem requires natural language processing (NLP) techniques that preserve the sequential relationships between words in each Tweet, making it suitable for a deep learning approach. I aim to compare results between different model types to highlight any improvements that the more complex deep learning methods bring. Using a baseline from traditional machine learning methods, including gradient boosted trees and multilayer perceptron models, I will also implement variants of recurrent neural networks (RNNs) and Transformer models. These deep learning models can learn the structural patterns in word order that traditional machine learning models discard, so I expect that they will perform better. However, while the RNN and Transformer models are more suitable for this NLP task in theory, it is important to show that this improvement is also attainable in practice. If my implementation of the deep learning models fails to give the expected improvements, I will also need to consider potential explanations for such results. Ultimately, this model for sentiment analysis can be a very useful tool for moderation on social media platforms. Filters that automatically detect hate speech or threats help to reduce negativity, and these could play a major role in reshaping online discourse to be less polarizing.

Problem Statement: I am using a dataset from Kaggle (link to dataset) for this project. There are a total of 40,000 observations, each with a Tweet ID, sentiment label, Tweet content, and Twitter handle of the author. The only useful variable for prediction is the text content of the Tweet, as most authors in the dataset are unique. There are 13 sentiment classes in the raw labels, but I may consider grouping some very small classes with similar larger ones. I will randomly partition the dataset into training/validation/test sets with a 60/20/20 ratio, allowing me to train and evaluate model performance at the intermediate stages using the training and validation sets, while also preserving a fully new test set to assess my final models and estimate their performance on unseen data. I will optimize categorical cross-entropy as the loss function during the model training process for this multiclass classification problem, and the most interpretable metric for assessing final model performance is classification accuracy.

Technical Approach: Overall, I am adopting much of the approach that was presented in the NLP task of HW2, and I will make changes in a few stages of the modeling process and compare results. Due to the computational costs of training these deep learning models, I am using the T4 GPU in Google Colab to run all of my code before saving the files locally and on GitHub (link to repository). While I have also created a local Python environment containing all of the required packages to run the notebooks, that is much slower and not my standard workflow.

One area of experimentation is the choice of an embedding model from SentenceTransformers. While HW2 used **all-MiniLM-L6-v2**, a simple yet effective option, there are many alternative pre-trained models that could give a performance improvement at the cost of added complexity and greater computational time. This step of generating embeddings is critical since it converts the raw textual data into the model inputs, so a change at this stage is more likely to impact model performance.

The more significant area of exploration is in comparing many different model types. My performance baseline will be the traditional machine learning methods that discard all sequential relationships in the text data, using a gradient boosted tree from XGBoost and a fully connected multilayer perceptron network in Keras. I will also implement Transformer models to give an upper threshold that I can aim to approach. However, the majority of my experimentation will be in developing different model architectures with variants of RNNs using Keras. For example, I will compare between vanilla RNNs, gated RNNs, and bidirectional RNNs, where each model type also has many submodels that differ in the number of layers, regularization, and other considerations. My ultimate goal is to identify a model that best learns the features of the data without overfitting to the training set, while also forming an understanding of why performance varies between the different models that I try.

Preliminary Results: I have completed the first steps of exploratory data analysis and running simple versions of each model type. As shown in **Fig. 1**, there is a potential issue of class imbalance in the target variable. In this already challenging multiclass classification problem, it will be extra difficult to handle classes with very few observations (<1% in anger, boredom), so a reasonable step could be to combine these with similar classes that are larger. However, I proceeded without making those changes in my initial testing.

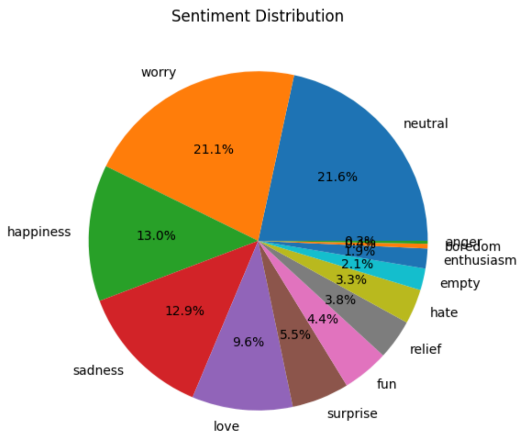

Figure 1. Distribution of sentiment classes in the dataset.

Using a nearly identical preprocessing setup to the NLP task from HW2, I defined a very simple variant of each model type and ran each for 10 epochs to ensure the fitting process would proceed smoothly with my dataset. The performance of all my preliminary models landed between 20-40% accuracy on the validation set after 10 epochs, with a cross-entropy loss between 1.8 and 2.2. While the vanilla RNN did not seem to learn over the 10 training epochs, all other model types were continuously improving and did not display signs of overfitting. Thus, while the current level of performance is suboptimal, these initial results show that my approach is reasonable and that it is definitely possible to develop a model that performs at a more desired level.

Next Steps: Having completed the initial testing, my next steps will be to proceed with exploration. It is easiest to begin by trying different embedding models and picking one to stick with to maintain consistency when comparing model types later. This could be done using only 1-2 model types, since any improvements from a different embedding should be similar across all models. Then, it would be reasonable to implement the baseline and Transformer models to identify a performance range for the other model types. With this general goal for how well to expect my custom models to perform, I will go through each model type separately and try different model architectures to identify ones that give the best performance without overfitting. Finally, I will assess my best models on the reserved test set and compare their performance to evaluate whether the deep learning approach was successful. In addition to the technical implementation, I will also maintain an organized GitHub repository and project structure so my results are accessible and easy to understand.