BMI 733 Final Report

**Introduction:**

The initial project was intended to classify posts on the Reddit subreddit r/lupus. The goal of this project was to apply what we have learned in class about text classification using logistic regression to identify posts that are popular or unpopular with the subreddit users; however, this changed to text classification using convolutional neural network (CNN). The subreddit sidebar describes r/lupus as a “[p]lace to connect, look for advice and exchange stories.” Since lupus is a relatively rare disease, those diagnosed with the illness, particularly those living in less populated areas, may not have access to in-person support groups. Forums like r/lupus can provide an online alternative that may help them in ways that their clinicians and other care providers cannot. These forums may also help those who are caring for friends and relatives with lupus, those who suspect they may have lupus, as well as those who have similarly rare autoimmune diseases. The team created an NLP classifier that can identify those discussion threads in r/lupus which could be adapted to similar forums that are most helpful to the various individuals who visit them, but “helpful” is a difficult concept to quantify. Fortunately, Reddit provides us with an admittedly imperfect proxy for helpfulness and unhelpfulness: popularity as indicated by the difference in upvotes to downvotes.

The stated purpose of an upvote, according to “Reddiquette,” is to show that “you think something contributes to conversation” (https://reddit.zendesk.com/hc/en-us/articles/205926439-Reddiquette). Conversely, a downvote indicates that a post or comment “does not contribute” or is not on topic in the context of that subreddit. Note that these “official” descriptions of Reddit upvotes and downvotes do not correspond precisely to agreement or disagreement, but in using Reddit one often encounters the widespread belief that upvotes are essentially “agreements” or “likes” and downvotes are “disagreements” or “dislikes.” For our purposes, the difference in the votes seems to indicate that the voter generally thinks the post or comment is popular or popular. But since we cannot know this for certain, and different users may upvote or downvote for different reasons, we will limit ourselves to classifying posts and comments as popular or unpopular. The Reddit algorithm promotes upvoted content and tends to hide downvoted content. According to subredditstats.com on October 17th, 2021, r/lupus averages 18 posts per day and 121 comments per day. A Reddit “post” is the beginning of a conversation and “comments” are the responses. Some posts are simply links to other web content or images, but we are interested in original text posts. We used all such text posts and comments from r/lupus for a several month period. Individual posts and comments are the “documents” used for training, tuning, and testing, and will have a score that is determined by the differences of the votes. The use of several month periods is intended to mitigate the potential volatility of total scores. We decided to divide the corpus into 80% training data and 20% test for training the CNN. We will use the same basic approach as in Project 3 to classify the posts, adjusting our filter sizes, and running 10 iterations of the model and taking the average of F1, precision and recall. Much of our work will be in developing useful features to increase the likelihood that our classifier will be able to achieve high precision and recall when applied to the final dataset. The “ground truth” popularity of the posts and comments in the final dataset will be determined by the differences in votes. Lastly, we looked at the validation loss vs. epochs, and validation F1 vs. epochs, to determine how fast the model learns.

**Methods:**

The methods used the CNN to model the popular and unpopular posts from r/lupus. The team used the following Python libraries, PSAW & PRAW for data retrieval that uses the Pushshift api to scrape data from the subreddit. This came with several issues as the two data retrieval libraries when using these scrapers, first issues that was making sure that the removed posts did not show up or posts that were empty did not populate the datasets. From there the comments combine with the self-text of the post were then scrubbed for anomality to protect user’s personal data for ethical reasons, along with URLs. Any posts containing only images were ignored for this project, since the focus of this study is to analyze textual anecdotes/recommendations rendered on social media by capturing the judgments made by others in the subreddit. This led to all the data being pushed into a csv to create the dataset. In total we scraped 40,050 unique posts for our training set, almost a whole year from November 1st, 2020, to October 31st, 2021. The datasets from there were then classified by using the top 80th percentile of the scores for the post giving us a value of 4, leading to 8010 “popular” posts for the training dataset. The Pandas library was used to aggregate the data from the csv format into data frames, from here the team used a Kaggle page to construct where the CNN would run on. Since, this was such a large dataset the team decided that the best method for running and compiling the code would to be using the TPU accelerators provided by Kaggle. The first steps taken by the team was to randomly split the training data by 20% for validation and 80% training. From here the data was tokenized using the Keras’ preprocessing text tokenizer and using Keras’ text\_to\_sequences to create the sequences of integers from the documents. After this previous step was to setup an word embedding matrix, the team decided to go with Stanford's GloVe 300d word embeddings public text file to generate the word embedding vectors for the matrix there came out to be 400,000 word vectors in this file. From here the word vectors were matched to the words within the training set to create our matrix. The next step in our project was to create the CNN model itself.

**Results:**

**Discussion:**

One of the main issues we had we had when developing our original model was determining how to incorporate a part of speech (POS) tagging for the original Logistic Regression model that was going to be used for the classification of post in r/lupus. Indirect use of POS was acceptable for our model as well; however, the team decided to change to a CNN after the third project, since neither of the members had prior experience developing neural networks and trying to figure out where the use of POS could be implemented in the model was difficult. Another issue with the logistic regression model that was found is that the model produced high recall values but very poor precision values, this could be since the original data being scraped from reddit was n = 716, as the new dataset was n = 40,050 posts, with a high variance in the new dataset the results were less than par; however, the smaller dataset did not give accurate scores for the whole subreddit but just a filter of new for the subreddit. Before fixing the dataset, the data ran on the CNN and was found to have better results F-scores closer to 0.72 whereas, before in the logistic regression the F-score was around 0.65. This led to the team producing a larger dataset to train/test the model, considering the dataset was quite smaller than the ones we used for the final model led to results not as great compared to that of the smaller one. This came with normalizing the dataset since the dataset was very unbalanced considering some post could be less than 100 words and be popular; whereas some post could be 400 words plus and be classified as not popular, and vice versa for the classification of other documents. This led to the team using the class weight parameter when fitting the model, to setting 0 to 1 and 1 to 2.47 allowing for a more equal distribution.

**Conclusion:**