Final Report: Classifying r/lupus posts & comments by popularity

Submitted by: Alex King
Jeremiah Davis
Course: BMI 733
Primary Supervisor: Ramakanth Kavuluru

Introduction

Our project intends to classify posts and comments from the Reddit subreddit r/lupus as "popular" or "unpopular." Our original goal was to apply what we have learned in class about text classification using logistic regression to classify these texts. Very early in our development and testing we saw that the performance of the logistic regression approach was so poor that this approach would probably not product satisfactory results even after adjustments and adding features. After gaining some experience in Project 3 using neural networks, we decided to use a convolutional neural network (CNN) based on the one from that project.

The lupus 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. Our team created an NLP classifier that attempts to 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. "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. This difference is known as the Reddit score. There is theoretically no maximum score, but for posts 0 is the minimum score. Comments may have negative scores, again with no theoretical minimum. By choosing a cutoff Reddit score as the dividing line between popular and unpopular, we turn the problem from predicting an integer to a simple binary classification. Note that we could have had human scorers determine whether they thought a post was "helpful" or "unhelpful," but we had neither the time nor funding for such an approach. We discovered that the model performed much better when a score of 2 (the median score for the entire dataset) or greater was used to classify a post as popular than when we used a score of 3 (corresponding to the upper quartile).

The stated purpose of an upvote, according to "Reddiquette," is to show that "you think something contributes to conversation" [3]. 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 score seems to indicate that the voter generally thinks the post or comment is helpful or unhelpful. 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 and comments. We used all such text posts and comments from r/lupus from November 2020 through October 2021. Individual posts and comments are the "documents" used for training, tuning, and testing, and will have a score that is determined by the difference between the total number of upvotes and the total number downvotes. The "ground truth" popularity of the posts and comments in the final dataset is determined by the scores. Our total dataset included about 40,000 such documents. Such a large initial dataset mitigates the potential volatility of total scores depending, for instance, on the time the comment or post was made. We divided the corpus randomly into 80% training data for the CNN and 20% test data, each with the same proportion of popular and unpopular documents. We used the same basic approach as in Project 3 to classify the posts, adjusting hyperparameters, and running 10 iterations on the final the model. We recorded the precision, recall, and F1-scores.

We did find any journal articles that were closely related to our topic. We did find one Master's thesis [4] and two student papers that were [5, 6]. In [4], Rohlin experiments with using term frequency inverse document frequency (TF-IDF) and Latent Dirichlet Allocation (LDA) to extract topics from Reddit posts. They also use a Naive Bayes classifier and support vector machine (SVM) to classify whether the posts are popular or unpopular based on their topics. The thesis describes how experimentation to obtain the best results attainable with this combination of topic identification and classification methods. Rohlin also discusses the difficult topic of how to

define popularity in this context. Several different methods are discussed such as the number of comments on a post, or number of views of a single post. One restriction that Rohlin puts on the dataset used in the thesis is that the score of the post must be at least 2 to show that the post had been voted on at least once. This helps eliminate some posts that may have not even been seen by other users since Reddit automatically "upvotes" once on behalf of the content creator. We might have used this restriction in our own data retrieval process, were it not for the fact that there is no way to tell if a post with a score of 1 has never been seen or simply upvoted twice and downvoted once (to give one of a countably infinite number of other possibilities). Rohlin sets the cutoff for classifying popular or unpopular posts at the 75th percentile. This had been suggested to us, and we ultimately did use similar approach of setting to cutoff to the score at the upper quartile for one set of 10 runs, as we will describe in the next section. Rohlin notes that internet "trolls" downvoting posts or generating off-topic posts for no reason tends to diminish the accuracy of classification models.

In [5], Segall and Zamoshchin investigate the popularity of posts on Reddit as a NLP problem. While most papers typically subject their datasets to a single subreddit or a small batch of them, Segall and his team ended up using a large amount of subreddits to obtain a dataset consisting of a random sampling of 2,046,401 posts. A Multi-Class Naïve Bayes Classifier (MCNB) was implemented in this experiment using two versions. The following list of features was investigated: Title, Domain, Subreddit, Over 18, Self, Author, Created, Thumbnail, and selftext. Like many papers on document classification the root mean squared error was used as a performance metric. Version A consisted of a full feature set. This led to the model to overfit due to the high variance in the data. To combat this, they also constructed Version B which had a reduced feature set, along with removing stop words or simple common words that occur amongst documents. One method that the paper proposed was ablative analysis, which is a process by which components are removed one at a time, to find out which feature is the least helpful. A common theme that occurs amongst these papers is that overfitting can occur quite easily since the data that comes from the Reddit has a high variance. The authors found that the subreddit feature showed the most success for the algorithm. They determined that the posts in the subreddits r/nsfw and r/atheism were the most "popular". The r/pic subreddit performed the worse. This may be due to the fact the better the image quality the more likely for it to be popular. The team discusses that stemming the words in their documents helped them get a reduction from -1.32% to 2.49% for the baseline classification error.

In [6], Shuaibi considers the question of the best model for determining popular posts from the subreddit r/askreddit. Some of the tools used in this paper were the same as ours, such as the use of the GloVe pretrained embeddings, though we did not find the paper until after our training and testing was complete. The author notes that the time of post occurred affected the popularity of the post. Some of the datasets exhibited high variance The models that were evaluated in this paper were logistic regression, K-Nearest-Neighbors Regression (KNN), and Random Forest Regression. The author employs the root square means (using standard Euclidean distance) and the squared residual values of each model to give common metrics for compatibility. The first model is the simplest out of the three and uses least squares to find a relationship between two variables, which is very useful for binary text classification. The KNN model works by computing and considering the examples that are closest in the feature space. The author states that in his research, "Linear regression was subject to high bias, KNN Regression is subject to having high variance" [6]. In this way he explains that there are often trade-offs when deciding model is best for classifying popularity. The next model, Random Forest Regression, works by bootstrap aggregation for random examples from the training set to fit the decision tree. Shuaibi found using word embeddings in a Random Forest Regression performs the best out of the models considered. One of our own team members thought of using a random forest model as well, but we lacked the requisite knowledge to implement it in the given time.

Two conference papers we found touch on our topic but are more distantly related. We did not consider link-based posts in our own research, but it is noted in [1] that link-based posts tend to receive a lot of attention, but that popularity may not actually be reflected in upvotes. Also discussed is the problem of posts with extremely high scores hiding new posts that probably *would* be popular. In [2], the authors explored how a post's title can influence popularity, regardless of content.

Methods

As noted above, we changed our approach from using a logistic regression based classifier to a CNN. There were several motivations for this. One was that we were unsure at the outset how to include part-of-speech tagging as a feature in the classifier. One team member also tried using a basic logic regression classifier early on the project on a very small dataset of recent posts, with only 716 documents, from r/lupus and found that without

any other changes a CNN performed better. After Project 3, we both had enough experience to move forward with the new approach, and we thought that adjusting hyperparameters would be a more efficient use of our time that coming up with new features and implementing them.

Our first step was to obtain a large, cleaned dataset appropriate for the classification task. We used the PRAW and PSAW libraries to scrape all the posts and comments from r/lupus from November 2020 through October 2021. PRAW works directly with the Reddit API and PSAW uses an external API called Pushshift.io. PRAW provides very detailed information about each post or comment, including most importantly for our purposes the post content and score. We also retrieved information about the time the content was initially created, whether it was given user awards (like "Reddit gold"), and whether posts had flair (tags generally indicating the post topic). We ended up not making use of the award and flair data. PSAW has more robust querying capabilities than PRAW, making use of an external database, and allowing us to retrieve every single post and comment in the date range, rather than just the top posts in some Reddit category like "new," "hot," etc. Fortunately, PSAW is designed so that it can be used in conjunction with PRAW, so we were able to make use of the best features of each. Our fetch function retrieves all comments and posts, but only keeps "self-text" posts and comments, meaning that we immediately discarded posts that only contained images, videos, or external links. Reddit posts may also be deleted by the user or removed by an administrator, but still appear in the search results, usually with the word "deleted" or "removed" in brackets. We also discarded all such posts, and any blank posts. We considered using a minimum text length for each document but ultimately did not do so. Some encouraging posts or comments that may be upvoted could be quite short, for instance, as could offensive ones, and on balance, especially when working with a CNN, it seems better to include more training data than less.

We cleaned this initial dataset in two steps. First, we removed the special Reddit markdown tagging that often appears in scraped Reddit data. To do so we used the redditcleaner library, which is designed for this task. Next, we turned our attention to preserving user privacy as much as possible. It is true that the authors of the data posted their posts and comments in a public forum where there is little expectation of privacy. Nevertheless, instructor comments on our initial helped us to appreciate the seriousness and necessity of respecting the authors' privacy as much as possible. Firstly, we did not even retrieve Reddit usernames in our initial query, so no posts or comments are directly linked to any document. We also made no attempt to determine who the individual posters and commenters were by indirect means, or in any way to connect different posts and comments by author. Despite all of this, it is possible some identifying information could be leaked. To lessen this possibility, we used a Reddit library called PrivacyFilter, written by Leo van der Meulen and available on his github page (https://github.com/lmeulen/PrivacyFilter). His code is available under the MIT license and so can be freely modified and used, even without attribution and for commercial projects. PrivacyFilter replaces identifying words like proper names and places with the token "<FILTERED>," optionally using the Spacy library for an NLP based filter. For our filter settings, we used this NLP filter, but had to make several changes. First, van der Meulen's code is focused on the Dutch language, so we had to change the Spacy pipeline to an English one, and modify few other sections of code for English language use. Dutch and English are very closely related historically, and this was a relatively easy process. Additionally, we did not want the filter to remove all numbers, medicines, or diseases from the text, so we commented out this portion of the functions. Lastly, we retain the initial category tags rather than changing them all to "<FILTERED". For example, in the original code both a person and organization would be replaced with "<FILTERED>" but in our dataset we have "<PERSON>" and "<ORG>" separately. This creates a richer dataset without compromising privacy. The filter does not work perfectly. Spot checking the data reveals that there are both false positives and false negatives in the cleaning process. Nevertheless, the vast majority of names, urls, and email addresses are removed, and this goes a long way in our goal of preserving user privacy. If we had had more time, we would have found or written a filter for removing Reddit usernames in the documents, but this did not occur to us until very late in the project when our testing was already complete. An unintended but useful effect of this Spacy-based filtering function is that contractions, which are so common in informal English language writing, are separated into separate "words." For example, "didn't" becomes two tokens: "did" and "n't". This is desirable because it gives our CNN more potential information for its filters. Also of note is the fact that this cleaning process keeps any emojis in the document text. In our project proposal, we suggested creating a feature for emojis. Making an explicit feature for this is not necessary when using the CNN approach, but it is good that we were able to retain this data for the CNN's filtering process. After scraping and cleaning, we had 40,050 unique posts in our total dataset. These documents were all created between November 1st, 2020, to October 31st, 2021, inclusive.

Next, we had to decide on a score cutoff for "popularity." This allows us to consider the problem as a binary classification one as in this semester's programming assignments. In our initial project proposal, we suggest that posts and comments with a score greater than 20 would be labeled "popular" and those with a lower score would be labeled "unpopular." After looking at the dataset, we found that it was unbalanced and this criterion for popularity would make it even more so. Just 627 documents or 1.6% of the 40,050 posts and comments have a score greater than or equal to 20. So, we then considered using the median or upper quartile scores as the cutoff. The median score is 2 and the score at the upper quartile score (the median score of the upper half as defined by the first median) is 3. In fact, so many documents are labeled 2 or 3 that this is slightly misleading. The score is 2 from quantile 0.39 to 0.66. The score is 3 from quantile 0.67 to 0.79. After a bit of experimentation using 4 as the cutoff, we decided to do two separate sets of 10 runs of the CNN, one using a definition of "popular" as greater than or equal to 3 (the upper quartile) and one using 2 (the median) in the same way. Our motivation for doing so was founded in the realization that the data was unbalanced either way. If we choose 2 as the cutoff, there are substantially more popular than unpopular documents. If we choose 3 as the cutoff, there are substantially more unpopular than unpopular documents. We wanted to see if tuning the CNN would have better performance with one or the other classification scheme, which, as we will see, it did.

We created two pairs of training and validation data using random sampling on an 80/20 split (each) of the popular and unpopular documents. As a result, the ratio of popular and unpopular posts was the same in both the training and validation sets. In both cases, the training dataset contained 32,040 documents (80%) and the testing dataset contained 8,009 documents (20%). This means that, in both cases, one document from the original set of 40,050 was "lost" somewhere in the process of sampling and recombining into the separate sets. We did not notice this bug until preparing the final report and did not have time diagnose the likely "one-off" error somewhere in the code. When the minimum score to classify a document as popular was 3, the training dataset contained 10,767 documents classified as popular (34%) and 21,273 documents classified as unpopular (66%). The testing dataset contained 2,691 documents classified as popular (34%) and 5,318 documents classified as unpopular (66%). When the minimum score to classify a document as popular was 2, the training dataset contained 19,799 documents classified as popular (62%) and 12,241 documents classified as unpopular (38%). The testing dataset contained 4,949 documents classified as popular (62%) and 3,060 documents classified as unpopular (38%).

We used a Kaggle Jupyter notebook for our CNN, which is essentially a fork and modification of the code provided to the class for Project 3. As in project 3, we used the TPU accelerator provided by Kaggle. This was all the more necessary given the large size of our input. We tokenized the text using Keras' preprocessing text tokenizer. We used Keras' text_to_sequences to create the sequences of integers from the documents. We used pretrained word embeddings, utilizing Stanford's GloVe 300d word embeddings public text file. In this way were able to generate the word embedding vectors for the matrix. There ended up being 400,000-word vectors in this file. From here the code matches the word vectors to the words within the training, thus creating our matrix.

Our primary evaluation metric was F1-score, and we also recorded precision and recall for each run. In the early stages of development, when we were still considering using a Reddit score of 4 (corresponding to quantile 0.8 for the whole dataset) as the cutoff for popularity, we attempted using the CNN with the above settings and everything else as in the "default" setting for Project 3. As one would likely expect, the results were not very good, but we were surprised to see *how* bad they were given the improvement seen when switching to the CNN. The validation F-score for a few runs was below 0.02.

First, we considered how we might "enhance" the text with some more relevant information. Doing so when using a CNN is almost like adding more features in logistic regression. In these initial runs, each "document" was only the body text, and for posts, the title prepended to the body. We realized that by comparing posts and comments, we were in some manner comparing apples to oranges, so we added the tag "post:" or "comment:" as appropriate to the beginning of the document. We also conjectured that the time of day a post is submitted is likely to affect voting patterns. For instance, if a subreddit is primarily used by users from North America, but a post is submitted by an Australian during normal daylight hours. Since we had the datetime string for each document's creation time, it was simple to also prepend the UTC time of creation to the document text. This immediately effective a modest improvement, but we were still frequently seeing F1-scores in the 0.10-0.20 range. Though we had information from our initial query about the post and comment awards, we decided not to use this. Gilding, in which a Reddit user awards another by giving them "Reddit gold" does often seem to correspond to a high vote count, and therefore popularity by the definition we are using. On the other hand,

awards are in some sense a different measure of popularity and we felt that this might confuse what our model was trying to achieve.

At this point we began experimenting with various alterations of the hyperparameters. This was for us a fairly laborious process involved much trial and error, though we were able to make use of much of what we learned in Project 3. The most improvement seemed to come from increasing the number of filters and the batch size. Eventually, we settled on 6400 filters and a batch size of 512. We reasoned that with such a large dataset and a diverse type of posts (some medically oriented, some personal and emotional; some quite long and others simple words of encouragement), we might need more filters to capture the nuance of differences. However, as we shall see this may have made our model too complex, contributing to overfitting. Nevertheless, these were the settings that appeared to result in the best F1-score even in the validation set. Since the model appeared to train slowel, we also increased patience to 100 and set the number of epochs to 300. We removed the maxlen setting when padding our training and validation sequences on the theory that we were perhaps losing valuable information from longer posts.

These changes all contributed to a dramatic improvement in the training metrics, often ending in F1-score well over 90% and a noticeable improvement by both of definitions of popularity. We will present these results in the next section, but at this point we can note that we already suspected that our hyperparameters were contributing to overfitting. This is because the training metrics continued noticeably to improve long after we saw no substantial changes in the validation metrics. We tried doing two things to mitigate this. One was to very slightly increase the dropout to 0.55. Secondly, as suggested in one Stack Exchange response by the user "desertnaut," we implemented class weights [7]. This is an attempt to balance out an "unbalanced" classification training set by weighting some classes differently in the loss function. We figured out how to use class weights with Keras, passing a class weight dictionary to one of the model.fit() keyword parameters. For our weights we used the simple ratio of our two classes ({0: 1.0, 1: 2.47} for the set in which there were fewer popular posts, {0: 1.62, 1: 1.0} for the set in which there were more). It seemed at first that this was reducing overfitting at least on some runs, but it is hard to say if it had much overall average effect.

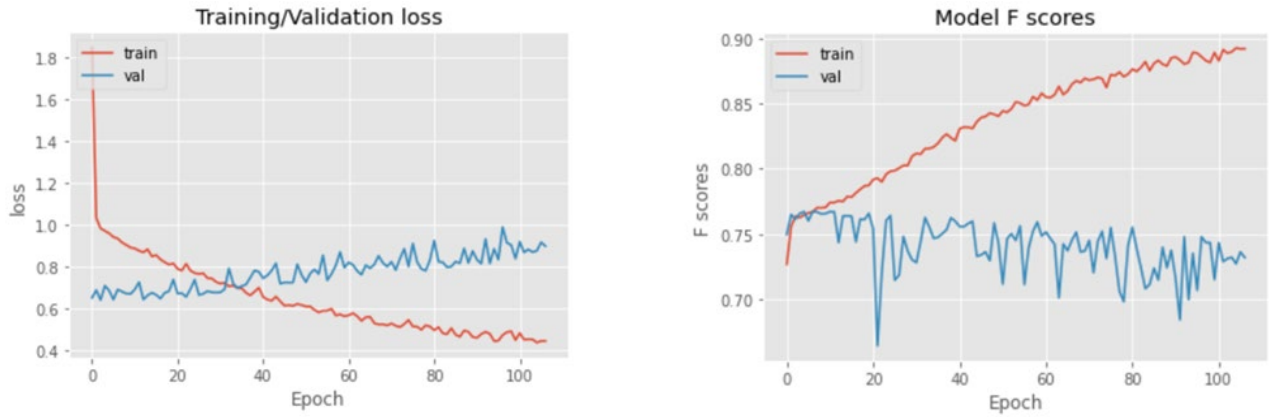
Results

Min. "popular" score	Avg P	Avg R	Avg F1
3	0.35	0.84	0.5
2	0.63	0.87	0.73

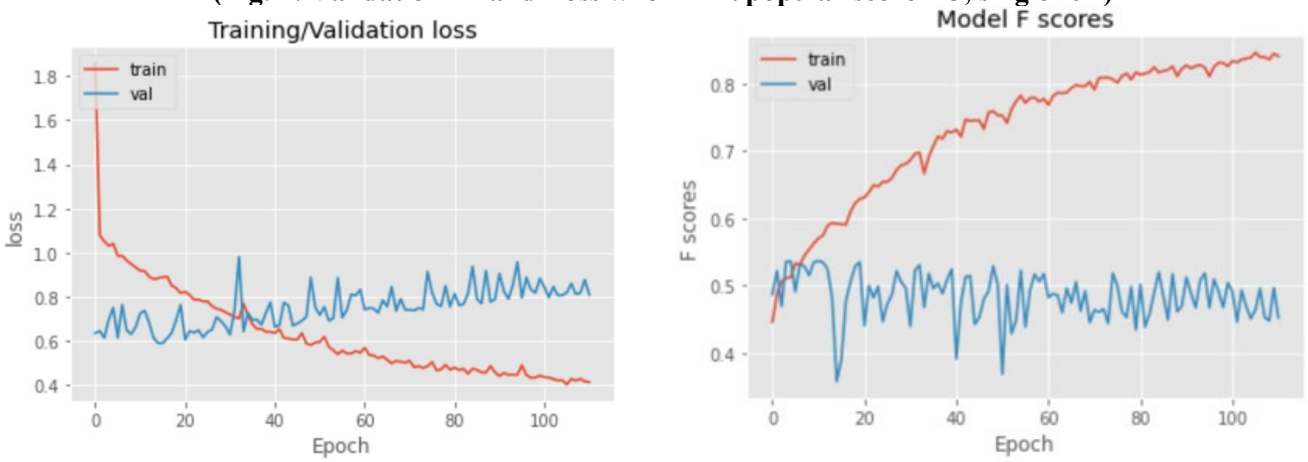
(Fig. 1: Validation Metrics)

In Fig. 1 above we display the average results over 10 runs with different our two cutoff points for popular scores. We found that lower cutoff results in a substantially higher F-score, and an almost 2 times increase in the precision. When the training dataset was set to 3 for the cutoff point, the difference in the number of popular documents (10,768) to unpopular (21,273) is quite large. This, along with the greatly differing lengths of the documents, results in high variance in the dataset. As described above, we attempted to "enhance" our documents to get more proper results, for instance by labeling the documents internally as posts and comments. We used the class weight parameter and attempting to regularize the data. Hand tuning several of the hyperparameters, such as the batch size and filter size, did help the team produce more accurate results. Having high recall and low precision was not ideal for the goal of the project. Having fewer false negatives is always desirable, of course. However, so many false positives means that we are often categorizing "unpopular" (and by proxy "unhelpful") posts as "popular" (by proxy, "helpful"). The second definition of popularity, where the cutoff is 2 or greater, probably improved the precision without substantially changing the recall because this adjustment helped to make the overall dataset more balanced (12,241 unpopular posts vs 19,800 popular). This means that when training the data, we had better class distribution. This may have even contributed to a "fairer" test/train split. It helped reduce the rate of false positives when analyzing the test set. The results from the second training set do compare favorably with what is seen in other papers, such as Rohlin's TF-IDF model on several subreddits, which produced F scores in the range of the high 0.60's and low 0.70's. However, her precision scores are slightly better, at about 0.1 higher [4]. The high variance in our particular dataset may, again, account for the slightly lower performance. The difference in the recall and precision scores for both cutoffs, leads to some concerns that the model may be overfitting since the recall values were quite high compared to the precision score. Also, as

previously noted, the overall training metrics were extremely high in comparison with the validation metrics, which further suggests ovefitting.



(Fig. 2: Validation F1 and Loss when min. popular score = 3, single run)



(Fig. 3: Validation F1 and Loss when min. popular score = 2, single run)

In the above figures, we see that the model learns at a slow rate. The main issue, however, is that the graph supports our suspicion that the model is overfitting when training the data. As seen in the Training/Validation loss graphs there is a large difference between the training and validation set before epoch 15. The rate at which the validation set's loss increases is much slower that of the training set, from the first epoch to the last one. Comparing across the 100 epochs the increase in validation loss was 0.2, whereas the training set begins at a large loss value close to 1.9 and ends with a value close to 0.45. Fig. 3 further supports the position that our model overfitting. In both figures, the rate at which the training F-score increases appears to be logarithmic. When looking at the model F-scores, the training F-score rate is more linear and shows no plateau while the validation F- scores tend to oscillate around the 0.50 or 0.70 range, respectively. This is another strong indication of overfitting. While the exact values of the F-scores differ depending on our cutoff value, their overall "shape" is quite similar. Another metric not shown here is the validation recall scores. Over the epochs for both sets, the validation recall scores tend to converge after the 80th epoch. Around the same epoch we can see that the oscillation of the F-scores becomes tighter in Fig 3. Later in the discussion section this will be addressed and how the team tried to mitigate this, along with methods we would implement later if time was allotted.

Discussion

We can say that in general that using a CNN for this purpose was modestly successful. We were able to achieve good recall scores for popularity as defined by upvotes equal to the upper quartile and median scores, respectively. For "median score popularity," we had respectable, if not fantastic, precision and consequent F1-score scores too. Unfortunately, our results also suggest that our initial intention of using NLP techniques to help those diagnosed with lupus and other rare diseases find "helpful" online content and avoid "unhelpful" content will require much more work.

The main, "human" problem is that having a high recall only means that the users of some future app, for instance, based on our method could be relatively confident that they are not *missing* much popular/helpful content. A low to moderate precision means that they will frequently get unpopular/unhelpful content recommended to them, which is precisely what they most likely do not want. Also, our model works best when "popular" is effectively defined as above average and even slightly below average! This is at best an idiosyncratic way of defining popularity.

As we have mentioned several times now, we suspect that overfitting is the main culprit here. One way we might fix this is to keep working on tuning the hyperparameters. As mentioned, we increased the dropout rate for the model from 0.5 to 0.55. This did appear to help somewhat, and dropout could probably be further adjusted with benefit. There seems to be a bit of "art" or intuition in tuning the hyperparameters, and even several more weeks working with the CNN would likely yield considerable improvement. Learning more about class weights and better alternatives to them might also help, given that some later reading on Stack Exchange after the initial post we mentioned tended to discourage their use. Yet another method to correct the overfitting would be to regularize the data to an even greater degree. For example, implementing a kernel_regularizer for the MLP layer in our model. This would apply penalties on the MLP layer parameters during optimization. Since this keeps the model less complex and works as a form of regularization there should be an improvement to the model. On the other hand, a new problem might arise; loss could be higher for the validation set.

Still another option would be to accept that there will be some "overfitting" and simply increase the size of our dataset accordingly. At one point we wanted to include all posts and comments from r/lupus from its beginning 11 years ago. Our scraping script makes this very simple. The only limitation is patience, because PSAW automatically backs off when it hits data transfer limits. As noted by our instructor, "overfitting" to extremely large datasets, such as those with millions of documents, can sometimes be desirable because you are essentially training your document on all possible types of documents in the field. We could also branch out to similar subreddits focusing on rare ailments and include them in our training dataset as well.

Including both posts and comments in the same dataset, even with in-document labeling, may have decreased the performance of our model. If as one of the background articles mentioned above suggests, there are different voting patterns for image posts, it may well be the case that the voting patterns for posts and comments is simply too different for direct comparison [1]. Especially if we were working with an even larger dataset as suggested in the previous paragraph, it might be worthwhile to attempt training two entirely different models, one for posts and one for comments. In hindsight, focusing especially posts may have been a better way of fulfilling our original purpose, because people may be more likely to want to see "helpful" original content than "helpful" responses.

We would also like to analyze individually those documents that most often appear as false negative and false positives. Unfortunately, we will have to run the models again to do so for the cutoff score of 3 because we neglected to keep this data for those runs. We realized our mistake when doing the cutoff=2 runs but unfortunately we ran out of time for even a cursory analysis.

Some other small changes we might try include our original idea of replacing emojis with their text descriptions. However, while this would almost certainly help if used as part of a feature is LR, whether this would actually help improve our CNN is questionable. We also had the thought of changing the basis for our word embeddings. Many posts have complex medication terminology, the word embeddings do not account for many of these out-of-vocabulary words. This could indirectly lead to some documents getting incorrectly classified. We might also include the flairs and awards in a post to help improve precision.

Conclusions

The use of CNNs for Reddit popularity classification seems like a promising avenue for further research. We have taken the first steps in creating a classification system that may in the future help real people diagnosed with lupus and other similar diseases find helpful online content from a variety of forums. We learned that the regularization of data is crucial for very unbalanced datasets, and that it is important to look out for signs of overfitting very early on in training and validation of a model. Both of us have limited experience in the field of machine learning, and it this was a valuable lesson in its particular challenges, which are quite different from implementing features in a LR classifier. The most important next step among those discussed in the previous section is probably to enlarge our dataset greatly. This may allow us to create a simplified model that is less prone to overfitting. A future extension of this project might be to program a post/comment that looks at a draft of a post before submitting it, determining if a post will likely be popular or not for the subreddit r/lupus or a similar medical subreddit. We both appreciate the opportunity to learn more about this topic and perhaps help others in the future.

CRediT Author Statement

Alex King: Initial scraper programming, preliminary testing and experimentation, CNN initial setup and tuning, Final paper writing. **Jeremiah Davis:** Advanced scraper programming, CNN adjustments and tuning, Final paper writing and editing.

References

[1] E. Gilbert. "Widespread underprovision on Reddit." *Proceedings of the 2013 conference on Computer supported cooperative work (CSCW '13)*. Association for Computing Machinery, New York, NY, USA, 803-808. doi: https://doi.org/10.1145/2441776.2441866

[2] H. Lakkaraju, J. McAuley, J. Leskovec. "What's in a name? Understanding the Interplay between Titles, Content, and Communities in Social Media." *AAAI International Conference on Weblogs and Social Media (ICWSM)*, 2013. url: https://cs.stanford.edu/people/jure/pubs/reddit-icwsm13.pdf.

[3] Reddiquette. https://reddit.zendesk.com/hc/en-us/articles/205926439-Reddiquette. Accessed: 2021-12-10.

[4] T. Rohlin. "Popularity Prediction of Reddit Texts". Master's thesis. San Jose State University, 2016. doi: https://doi.org/10.31979/etd.d7nw-6gx7. url: https://scholarworks.sjsu.edu/etd_theses/4704.

[5] J. Segall and A. Zamoshchin. "Predicting Reddit Post Popularity". Student paper. Stanford University, 2012. url: http://cs229.stanford.edu/proj2012/ZamoshchinSegall-PredictingRedditPostPopularity.pdf.

[6] A. Shuaibi. "Predicting the Popularity of Reddit Posts". Student paper. Stanford University, 2019. url: http://cs229.stanford.edu/proj2019spr/report/42.pdf.

[7] https://datascience.stackexchange.com/a/13496.