Reddit user classifier

David Chu, Jasper Duan, Richanshu Jha

[1]Practical NLP Project Report

dfc296@nyu.edu, zd793@nyu.edu, rj1469@nyu.edu

Abstract. *A problem in Natural Language Processing (NLP) research is accumulating large training data corpus. Traditionally this involved costly methods such as using humans to manually categorize texts. However, with the rise of social media, we now have access to large amounts of semi-structured text. This allows for new methodologies for collecting and analyzing text. One such social media site, www.reddit.com, is especially useful because its members self segregate into groups (subreddits), which can act labels. In this project we empirically explore how best to use Reddit data as input to a classifier by trying different pre-processing steps and models to maximize classification accuracy.*

1. Introduction and Motivation

There is a rich amount of data from people expressing their thoughts on social media. Humans can understand what kind of a person is from these expressions, but is there some way to this learn this through an algorithm? Using NLP, we would like to be able to identify what subreddit affiliations a user on the social website Reddit has (ie: /r/democrat or /r/republican) based on an NLP analysis of their post history. Our goal will be to generate a vector of values indicating how strongly associated a user's history is with each subreddit. Because a subreddit is focused around a common interest, this program will provide us a basis with which we can identify what a general person may be interested in based on his or her writings.

2. Problem Statement

The underlying idea behind this project is that the grammar, vocabulary, topics, and comment patterns are different across various subreddits based on the target demographics of the subreddits. With a large enough dataset, we can train a classifier to identify these variations in the comments and associate the comment patterns of the user to associate them to a subreddit and hence, that demographic. The scope of this project includes sets of subreddits that may be targeted to a specific, mutually exclusive demographics; for this we have initially selected $r/democrats$ and $r/republicans$. Once we have trained our model on these subreddits, we will attempt to infer information given a corpus of a user's entire comment history.

3. Related Research

Research have been done on Reddit users, but the majority of classification research are motivated by a goal of identifying certain characteristics about the user. For example, Gjurković et al. [2020] one research paper attempted to identify a user's demographics such as age and race and encountered issues with a "lack of datasets with both personality and demographic labels". Skowronski [2019] Another paper, motivated by concurrent

issues of Russian trolls, attempted to identify trolls and bots by training against known labeled bot and troll accounts on Reddit. Dyagilev [2019] Another related research attempted to classify users according the Myer Briggs personality types. This paper focused on using "very introspective" posts as the training set and used Reddit more as a basis for identifying how the Myer Briggs personalities are manifested in online writings. Although these research are very similar, our project focuses more on the subreddit origin of the posts themselves, and any inferences are made based on what the subreddit's topic is.

4. Dataset

Reddit text data can be thought of as a tree structure. Each "subreddit" can be thought of as the root node of that tree. The children node at the first layer are "submissions." These submissions consist of a title, and either text or a hyperlink. Submissions can have children, called "comments", which are replies to the submission. Each comment can then have it's own comments in reply.

Our dataset was taken from two subreddits, $r/Democrats$ and $r/Republican$. The r/Democrats subreddit has about 158,000 users and 600,000 comments as of 11/27/20. The r/Republican subreddit has about 163,000 users and 690,000 comments as of 11/27/20.

From these two subreddits, we retrieved all comments. We chose to use comments rather than submissions due to the fact that text submissions are quite rare in these two subreddits. We then filtered the comments using several criteria:

The first criteria we used was to remove comments that were deleted or removed (indicated by the text being "[deleted]" or "[removed]"). Without any textual data, these comments were not useful for our NLP models.

The second criteria we used was to remove comments that had a score less than a certain threshold. Reddit scores are determined by users. Upon creating a post or comment, that comment is given a score of 1. Each user can either add (upvote) or subtract (downvote) one point to the score of a comment. Furthermore, each reply to a comment also increases the score by one. We make the assumption that there is a positive correlation between the magnitude of a comment's score and how representative that comment is of the subreddit's users. We began with a score threshold of one, meaning that we keep any comment that has at least been upvoted as many times as it has been downvoted. We then experimented with increasing the threshold for both training and validation sets and for only training sets.

This left us with 653,049 comments. 342,511 Democrat comments and 310,538 Republican comments.

4.1. Comment Scores

As part of our experiments, we increased the comment score threshold from 1 to 7 points. We did this with a hypothesis that higher scored comments should be more representative of each other than lower scored comments, and so training on high scoring comments should yield better accuracy.

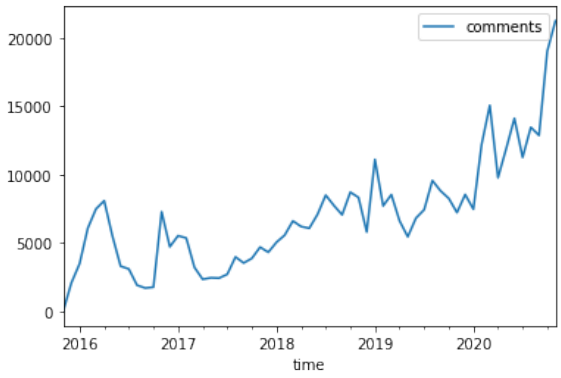

Figure 1. r/democrats comments 2016-Present day

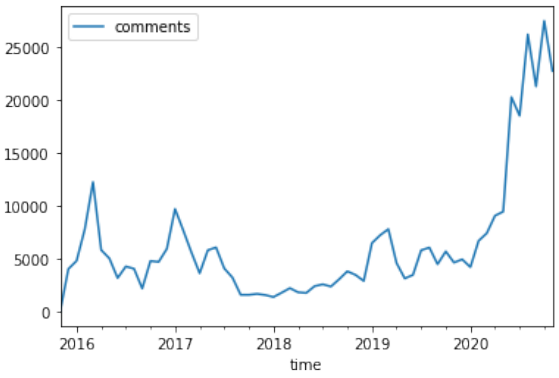

Figure 2. r/republican comments 2016-Present day

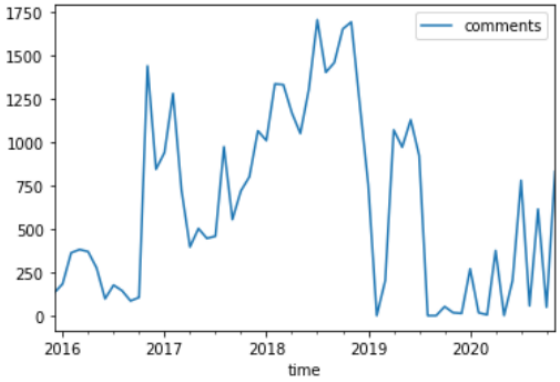

Figure 3. r/democrats comments with score >= 7, 2016-Present day

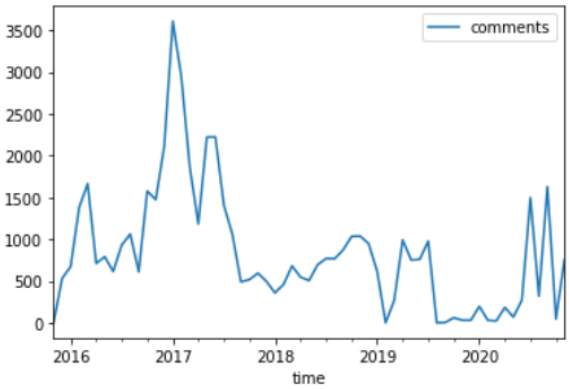

Figure 4. r/republican comments with score >= 7, 2016-Present day

5. Methodology

We have developed a pipeline that takes the raw extracted dataset in the form of pickle (.pkl) files and performs a set of processing, training and evaluation steps to create the classifier. The pipeline is fully configurable in terms of the choice of preprocessing steps, models, and evaluation methods. Each of the configurable steps in the pipeline are detailed below.

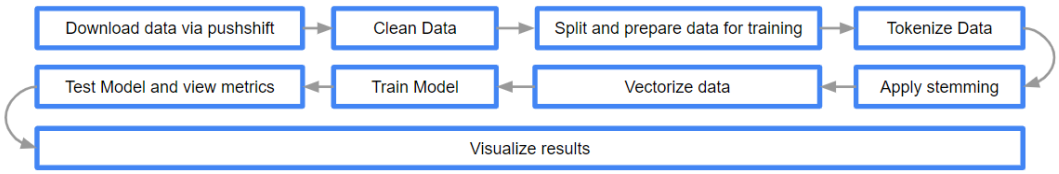

Figure 5. The data pipeline

5.1. Preprocessing

5.1.1. Tokenization

Tokenization is the process of splitting text into smaller pieces. For our project, we used the NLTK library to do word tokenization. NLTK offers two default word tokenizers, WordPunct and Treebank, that differ in how they treat splitting on punctuation. Given a word "won't", WordPunct will tokenize it into three pieces, "won", "'", and "t". Treebank will tokenize it into two pieces, "wo" and "n't".

The word tokenizer also had optional use of removing stop words contained in nltk.corpus.stopwords.words('english'). Stopwords are common words that generally do not aid in NLP tasks, such as "the" and "or".We also used a sentence tokenization using a custom sentence tokenizer which split on "." and used to split each data point (comment) into multiple data points; the class for each sentence is set as to the class of the comment it was split from.

5.1.2. Lemmatization and Stemming

Lemmatization and stemming are useful way to reduce dimensionality of our word vectors without sacrificing meaning. Lemmatization is the process of reducing different inflected forms of the same word into a single token. Lemmatization algorithms take into account contextual information such as part of speech and surrounding words to intelligently reduce. Stemming is a simpler method that does not use contextual information and simply replaces each word with its root form without contextual information.

We used the NLTK library to perform lemmatization and stemming. The NLTK library offers several algorithms, including Porter, Lancaster, and a WordNet based algorithms.

5.1.3. Vectorization

Natural language processing requires the tokens to be encoded in a way that is in a format that can be processed by Machine Learning models. Encoding these sentences into vectors of, in our case, floating point numbers is termed as Vectorization. In this project, we test the various configurations of the pipeline with 3 different types of Vectorization techniques: Count Vectorization, TF-IDF Vectorization, and Latent Dirichlet Allocation (LDA) Vectorization.

Count Vectorization requires the generation of a vector of the size of the vocabulary where each token (In our case: word) maps to a specific index in the final vector. This vector is generated for each document. The value associated with that index is the frequency of that token in the document. Count vectorization was chosen as a baseline vectorization technique over the one-hot encoded Bag of words representation; TF-IDF was selected after testing with count vectorization. TF-IDF Vectorization similarly requires the generation of such a vector, but the values with each index are the TF-IDF value associated with that document. This is the product of the term frequency tf and the

inverse document frequency idf for term t in document d in the set of documents D.

$$tf(t, D) = log(1 + freq(t, d))$$

$$idf(t, D) = log\left(\frac{N}{count(d \in D : t \in d)}\right)$$

There are many words that are common across both subreddits. These words such as 'politics', 'left/right', etc. may not necessarily be stop-words but occur almost equally frequently in both subreddits. TF-IDF Ensures that these words get lesser weightage due to their high document frequency.

Latent Dirichlet Allocation (LDA) is a generative statistical model that attempts to perform unsupervised classification of documents into a predefined number of topics. The result of LDA Vectorization for a document is a vector of the size equal to the predetermined number of topics, with a floating point score to each topic as the values. In this form of vectorization, the documents encodings are not a direct encoding of the tokens themselves, but are a representation of the distribution of topics generated for each document. Subreddits that cater to mutually exclusive demographics tend to have a difference in the topics of conversation; topic distribution vectors can thus be provided to the model to attempt to classify the comments based on the subreddit, hence LDA was select as one of the vectorization techniques in the pipeline.

The number of splits i.e. the number of topics considered is critical to the classification process and hence we tried to optimize this before further testing. LDA Vectors were created incrementally by using various splits and it was seen that increasing the number of splits increased the time and memory requirements of the training pipeline, limiting us to 500 topics. The generated LDA vectors were then used for training, all other parameters of the pipeline remaining the same. It was observed that increasing the splits increased the accuracy; however this increase became negligible after 150 splits. For the purposes of tuning the pipeline, 25 was chosen as the final topic vector size as it provides considerable accuracy with smaller memory and time requirement.

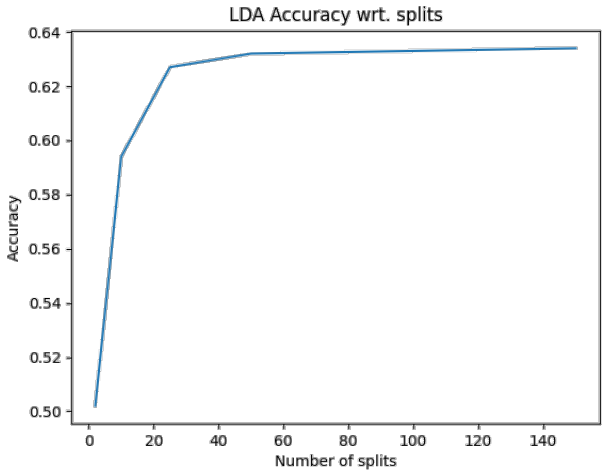

Figure 6. Accuracy of the pipeline configuration with different splits

Count Vectorizer, TF-IDF and LDA implementations used in this project are from Scikit-learn's features extraction module which use sparse matrices to optimally store vectors. Count vectorization has been performed as a preprocessing step for the LDA Vectorizer.

5.1.4. Data Loader

The pipeline has been tested and tuned on devices with 16 GB available memory. This allowed testing for Logistic Regression, Naive Bayes, BERT (On Colab), and SVM. However the model training failed for our neural network which is detailed in the $Model$ section of this paper. Due to the complexity of the network and the size of the vectorized dataset, the memory requirement for this model exceeded the memory capacities for our devices. The total required memory for the tested stemmers using count vectorizer and our model is provided below.

Stemmer	Vector Size	Memory Requirement
Porter Stemmer	34379	29.9 GB
Lancaster Stemmer	29799	25.9 GB
Lemmatizer	46374	40.2 GB

Initial training for the neural network was done using a 0.25 fraction of the dataset. These tests showed comparable results to tests with Naive Bayes and Logisitic Regression with the same sample. In order to overcome the memory issue and train with the full dataset, we have developed our own data loader for the pipeline specifically for the neural network. In this data loader, the entire raw dataset is loaded onto the main memory. We opted to load the entire dataset since this raw data can be accomodated in the memory; loading partial data at each batch would result in much slower training time caused by the high number of file reads.

The loaded dataset is used to fit the vectorizer but no transformations are done before training. For each batch in the model training, the data generator generates the next batch of data by accessing the subset of data required from the memory and returning the transformed (vectorized) batch to the model. The data loader stores only the order of indices of the data to be called. This order is shuffled at the end of each epoch. The data loader has been made by using Keras's $Sequence$ class, which is readily accepted internally by Keras's $model.fit$ function. The neural network has been trained with the full dataset using the data loader. This trained model is termed as NN-Optim in the $Results$ section.

5.2. Models

5.2.1. Naive Bayes

Naive Bayes is a probabilistic classifier that is based off of the Bayes Theorem in mathematics. It works on the idea that the posterior probability is the product of the prior probability and the likelihood, divided by the evidence. We chose to start testing with Naive Bayes as the baseline model. The advantage of working with this model is that it works well with the baseline vectorizer (Count vectorizer) and it is also computationally

inexpensive, allowing other parameters of the pipeline to be tested and optimized. We have used Scikit-learn's implementation of Naive Bayes for this project.

5.2.2. Support Vector Machine

Support Vector Machine is a statistical classifier that is used for prediction. As a supervised learning model, it tries to find the best hyperplane that represents the largest separation, or margin, between classes. Due to how the SVM calculates a hyperplane, the model is inefficient for learning on large feature sets where it would have many parameters to tune.

5.2.3. Logistic Regression

Logistic Regression (LR) is a statistical model that is extensively used in the creation of categorical classifiers. Given that our the task is to classify comments into two distinct classes, Logistic Regression is well suited for this task and was selected after testing with Naive Bayes. The pipeline trains an LR model using Sci-kit Learn's implementation; the only change in configuration being that the maximum iterations have been increased from 100 to 1000.

5.2.4. Deep Neural Network

Neural Networks (NNs) or Artificial Neural Networks (ANNs) are a network of mathematical functions that contain weights and biases called neurons. Each neuron has its own weight and bias, takes input from one or more other neurons in the network and feeds the output to one or more other neurons in the network. In this project, we use a layered feed-forward dense network structure where the network is sectioned into layers such that each neuron of a layer forwards its output to all other neurons in the next layer.

The architecture of our model is a deep neural network where the input layer has a neuron for each element in the vectorized document. There are 3 hidden layers with 512, 256, and 128 neurons, each having a Rectified Linear Unit (ReLU) activation function. The final layer has 2 neurons with soft-max activation. The total number of neurons depend on the input vectors supplied by the vectorizer, for an input vector of size 10, there are a total of 5,284,994 trainable parameters. Tensflow has been used along with Keras's Sequential module for the implementation of this model. The complete architecture is tabulated below.

Layer	Output Shape	Activation	Parameters
Dense	(*,512)	ReLU	5120512
Dense	(*,256)	ReLU	131328
Dense	(*,128)	ReLU	32896
Dense	(*,2)	ReLU	258

5.2.5. BERT-based Neural Network

Bert based neural network for classification is composed of a set of Bert encoders and a single layer feed-forward neural network. The encoders transform the text input into vectors that are then fed into our feed-forward neural network to classify the text.

For our project, we used the BERT standard implementation provided by huggingface.co, which uses 12 encoder layers. We used the pretrained model for classifcation, only fine-tuning the feed-forward neural network for our classification task. We used softmax cost as our loss function. Each input to our model is a single comment, tokenized using huggingface.co's pretrained BertTokenizer.

We also attempted using a custom loss function to weight the softmax cost by the reddit score of the text. The intuition behind this was rather than reduce the amount of data by filtering out comments, we could instead keep the data, but weight it proportionally to its score. In this way our model could still learn from low scoring comments, but would not learn as much as from high scoring comments. Our assumption being that low scoring comments are not as representative as high scoring comments, and so should not be weighted the same.

The standard cross-entropy loss is:

$$CE(\mathbf{w}) = -\sum_{p=1}^{P} \log\left(1 + e^{-y_p x_p w}\right)$$

A weighted cross-entropy loss might look like:

$$CE(\mathbf{w}) = -\sum_{p=1}^{P} \beta_p \log\left(1 + e^{-y_p x_p w}\right)$$
$$\text{where } \beta_p = f(x_p^{score})$$
$$\text{and } 0 \leq \beta_p \leq 1$$

5.3. Evaluation

In order to evaluate our models' performances, we set aside a portion of our dataset as the validation set. This labeled validation set is used to measure the accuracy, precision, recall, and f1 score of the model predictions. In order to get comparable results, we ensured that each configuration of the pipeline is being evaluated on the same dataset or same sample of the dataset.

In order to test the models with live data, we developed a visualizer module. The pipeline can be configured to save the model and vectorizer to storage after they have been fit. The visualizer module works with any set of model and vectorizer saved into storage. It takes as input, a username or comment, uses pushshift APIs to fetch all comments of the user, and performs predictions on all comments that are posted to political subreddits. Of all the 'political' comments made be the user, the comments and predictions to $r/democrats$ and $r/republican$ are separated out and a validation accuracy is calculated for these comments. This module is made such that it can be hosted on a local flask server and has a web interface to view the results of each classification.

6. Results

We have performed various tests by configuring the parameters of the pipeline. While work on optimization and testing of all of the modules is ongoing, the current results of testing are mentioned below, sorted by increasing order of accuracy. These tests have been performed with the data-set as described in the $Dataset$ section of this paper.

Tokenizer	Stemmer	Vectorizer	Model	Precision	Recall	F1	Accuracy
WordPunct	Lemmatizer	LDA-5	NB	0.432	0.564	0.489	0.553
WordPunct	Lemmatizer	LDA-5	LR	0.513	0.557	0.534	0.557
WordPunct	Lemmatizer	LDA-5	LR	0.729	0.578	0.645	0.602
WordPunct	Lemmatizer	LDA-10	NN	0.620	0.577	0.598	0.615
WordPunct	Lemmatizer	LDA-25	LR	0.538	0.644	0.586	0.624
WordPunct-SS	Porter	TF-IDF	NB	0.629	0.664	0.646	0.672
WordPunct	Porter	Count	NB	0.700	0.672	0.686	0.682
WordPunct	Lancaster	TF-IDF	NB	0.691	0.682	0.686	0.687
WordPunct	Porter	TF-IDF	NB	0.704	0.684	0.694	0.693
WordPunct	Lemmatizer	TF-IDF	NB	0.711	0.686	0.698	0.696
WordPunct	Porter	TF-IDF	SVM	0.696	0.709	0.702	0.697
WordPunct	Lemmatizer	TF-IDF	SVM	0.697	0.707	0.702	0.697
WordPunct	Lemmatizer	TF-IDF	LR	0.702	0.690	0.696	0.701

The number after LDA signifies the number of splits and '-SS' after Tokenizer denotes that the document has been split into sentences. LR,NB,NN are acronyms for Logistic Regression, Naive Bayes and Neural Networks respectively.The Precision, Recall and F1 score in the table are with respect to class 0.

A Neural network was trained was trained with the full dataset without the data loader. This was made possible by using LDA vectorization since memory requirement greatly reduces as the vectorized data shape reduces from (*, 40000) to (*,25). Its results are available in the table above. The neural network as trained using the data loader was tested with various hyperparameters. The results of these tests are available below. WordPunct was used as the tokenizer for all tests.

Stemmer	Vectorizer	Batch Size	Epochs	Precision	Recall	F1	Accuracy
Lemmatize	TF-IDF	2048	2	0.683	0.700	0.691	690
Porter	TF-IDF	2048	7	0.690	0.682	0.686	691
Lemmatize	Count	2048	4	0.687	0.697	0.692	693
Lemmatize	TF-IDF	4096	2	0.700	0.667	0.683	694

As detailed in the $Models$ section, we have also trained a BERT Model. Huggingface's transformers are being used on the data and hence its implementation is separate from our pre-existing pipeline. The trained model's accuracy metrics are provided below.

Sequence Size	Model	Precision	Recall	F1	Accuracy
256	BERT	0.674	0.793	0.728	0.708

6.1. Comment Scores and Date

We also ran several experiments filtering the data by score and date of creation.

Train Set	Validation Set	Accuracy
Full	Full	0.68
Score >= 7	Full	0.62
Score >= 7	Score >= 7	0.68
Created in 2020	Created in 2020	0.69
Created in 2020	Created before 2020	0.60

7. Discussion

7.1. Machine learning algorithm performances

The best performing model was our BERT model with an accuracy of 0.708. However, our second best model was consistently logistic regression, with an accuracy of 0.701. This was surprising given neither Naive Bayes nor logistic regression take into account positional data about words in the text. Despite this, BERT was still unable to achieve a significant accuracy improvement. This accuracy of around 70 percent seemed to be an upper limit to what we were able to achieve. By looking at the misclassified examples, we can hypothesize about why this might be the case.

7.2. Misclassification

Below we list some misclassified examples when using our trained Logistic Regression model. When looking at sample comments, there are several reasons why they might be misclassified. Several reasons might be:

1. Democrats and Republicans will naturally have some overlap in opinion. Political parties do not necessarily divide people cleanly. Some Democrats might have republican-like opinions and vice versa. For example, a Democrat could use the phrase "BLM rioters," despite it being more common among Republicans.
2. Some comments may be agnostic to political leaning. For example a short comment such as "Kasich?" would be difficult to classify without any context. However, we did not find that there was a correlation between comment length and classification accuracy. Another example might be a comment that is simply stating a fact, such as "That's two less votes for Bernie Sanders."
3. Models that only rely on word counts may not be able to understand concepts like sarcasm, questions, etc... For example, "Why are antifa and BLM bad organizations?" is strongly classified as Republican, indicating our model doesn't understand how a question can be different than a statement.
4. Lack of context may make entity recognition difficult. For example, a comment such as "I still don't understand why anyone is acting like this guy is a human when he would throw his own kids in front of a bus to save himself." Without context its not clear who this sentence may be talking about, leading to a low-confidence label.

7.2.1. Democrat comments misclassified as Republican

Comment	Prediction Probability
Kasich?	0.99
Said a bunch of BLM rioters	0.99
Why are antifa and BLM bad organizations?	0.99
But Kasich won the primary in the state he was governor.	0.99
...	...
What took you so long?	0.5
So its skewered to say what we want to hear.... cool.	0.5
I still don't understand why anyone is acting like this guy is a human when he would throw his own kids in front of a bus to save himself.	0.5
That's fair. I'm not out here picking fights with you man. I just have a different outlook than you lol. In my mind, we're winning in the polls, were crushing him at debates, we don't need to start gossip.	0.5
I wonder how "his" Police, military and bikers feel about working for Russia, these are the types that believe his BS, and one is all it takes to turn on him, if they start to wise up that Trump is lying to them.	0.5

7.2.2. Republican comments misclassified as Democrat

Comment	Prediction Probability
But look at who else they voted for. Amy Klobuchar, Pete Buttigieg, and Joe Biden. All three of those are much more moderate than Sanders. At most he may get Biden's and Buttigieg's, but from what I've seen Klobuchar supporters are firmly against Bernie. That puts him at just over 50%	0.99
That's two less votes for Bernie Sanders...!	0.99
I would assume not so much Sanders, because he is getting old, but Warren and Booker will be the new leaders of the DNC.	0.99
Republicans do not let other Republicans support Bernie Sanders.	0.99
They're not going weaken their power base for Bernie Sanders	0.99
...	...
No. Donald Trump acknowledges that America has bargaining power it is not properly utilizing. Free trade has to go both ways to properly benefit both parties and right now it is not. If other countries won't negotiate he'll bring down the hammer.	0.5
I just rolled my eyes so hard the guy in the next office was able to hear it.	0.5
didn't we see 6 years of republican congressional resources dedicated to attacking Clinton? Both sides do it.	0.5
Are they just trying to give us reasons to get mad at them?	0.5

7.3. Comment Scores and Date

We had surmised that the comment score is a better representation of a subreddit than a comment scored lower, because they have more validation from that subreddit community. However, our investigation reveals that this feature does not improve accuracy of predictions. Training using high scored comments, does not lead to increased accuracy for either the full validation set or a validation set of similarly scored comments. Similarly attempting to use a weighted loss function does not have any effect on accuracy. This suggests to us that the lower scored comments are a significant contribution to the subreddit's representation when doing predictive classification. Furthermore it seems to imply that highly scored comments are not more linguistically similar to each other than lower scored comments.

We also hypothesized that subreddits' contents change over time. We did observe a drastic drop in prediction accuracy when we trained the model on past comments and had it predict comments posted at a future date. This shows that there are new content which need to be learned. The overall prediction improvement is marginal when limiting both training and validation sets by date, which suggests that we are still near the peak prediction accuracy of our dataset even with the addition of the date feature.

8. Conclusion

8.1. Summary

In doing this research, we had hoped to find a strong correlation between a subreddit and its posts that we can then leverage to identify a user's post. By using different NLP models, we discovered that there is a significant correlation between the content of a user's posts and their subreddit tendencies.

However, in our investigation, we also realized many oversights. Mainly, that not all posts can be classified with just the text of the post itself. Because there are other things such as the context, sarcasm, and neutral/factual posts, our classifiers peaked around 70% accuracy. We also saw many neutral posts in the subreddit corpus, but did not originally think of a way to handle this class.

8.2. Limitations and Further Research

As seen in our results, we were reaching the upper limit of what we could achieve in prediction accuracy. We do not believe this to be a limitation of our classifier, because we tried many permutations of different types of classifiers. We believe this limitation is due to either (a.) the nature of posts and comments being specific to the thread it is in or (b.) the posts are agnostic to the subreddit or are "neutral". However, this would be different from our original goal of having a general classifier that can map any text (not a post in a thread) to a subreddit.

There is potential for further research into creating more robust models based on post context and additional features. However, this brings up the question of what to do when there is no context?

Another extension to our research would be to try multi-class models that would provide probabilities for a post to appear in different subreddits. This probability distribution can then be studied. For example, certain subreddit likelihood distributions can potentially reveal similarities and overlaps between users or subreddits.

References

Vladimir Dyagilev. Using deep learning to classify a reddit user by their myers-briggs (mbti) personality type, Jul 2019. URL https://medium.com/swlh/6b1b163194d.

Matej Gjurković, Mladen Karan, Iva Vukojević, Mihaela Bošnjak, and Jan Šnajder. Pandora talks: Personality and demographics on reddit, Apr 2020. URL https://arxiv.org/abs/2004.04460.

Jason Skowronski. Identifying trolls and bots on reddit with machine learning (part 2), Jul 2019. URL https://towardsdatascience.com/709da5970af1.