Rahul Malhotra
Vikrant Kumar Chandan

#Classifier Final Report

Introduction

For our project the goal was to design multinomial classifiers with the task of predicting which hashtag a tweet belonged to, and determine which of these classifiers performs the best. Since tweets are not limited to a single hashtag and there can be multiple popular hashtags for a given topic, it was important to pick hashtags that were not too closely related to one another. Otherwise, our model would struggle to distinguish between which hashtag to classify the tweet as. The four hashtags we decided to use were “crypto”, “thanksgiving”, “lockdown”, and “dogs”.

After collecting and cleaning the data, we split our entire dataset into a training and test set using an 80-20 split. Then, this training set was further split into a smaller training set and a validation set, again using an 80-20 split. The next step was to determine what types of models we were going to implement, as well as what metrics we were going to use in order to compare them. For the baseline models, we decided to use logistic regression, as well as Naive Bayes, and for the neural network models we used two different architectures, each with two different input embeddings (TF-IDF and GloVe). Once the models were built, we used both heatmaps and F1 scores to compare them. The heatmap of each model provided a more visual comparison and qualitative understanding of how a model was doing, while the F1 scores (mainly micro-averaged since it serves as a better metric in comparison to macro-averaged when the data is imbalanced) were a quantitative measure and ultimately served as the deciding criterion for the best model. Our analysis found that the neural network model using TF-IDF inputs, 2 hidden layers, and 100 units each performed the best with micro and macro F1 scores of 0.9423 and 0.8853, respectively. The worst performing model was the neural network using GloVe inputs, 3 hidden layers, and 50 units each with micro and macro F1 scores of 0.8116 and 0.6078, respectively.

Data Collection

Once we had a conceptual and initial understanding of what we wanted to accomplish in this project, it was time to actually collect some data that we could work with. Using python’s “tweepy” library and the code provided, Rahul set up our listener and began collecting tweets from about November 20th to November 27th. To start, different hashtags were used and the rate at which tweets were being collected for each was recorded. Since it is important to have sufficient data for our hashtags, we had to make sure that the hashtags we were going to use were popular enough to collect a large number of tweets within the given time period of the project. After trying out several hashtags, we decided on the four mentioned previously and began writing the tweets into files, where each hashtag had its own

corresponding file. However, while we did pick the more popular hashtags, we later found in the cleaning process that a majority of the tweets were being filtered, thus we had to run the listener for longer and collect more tweets. As a way to add to the collection of tweets so far for a particular hashtag, we made sure to append to a hashtag's file, rather than write to it, otherwise the previous data would be overwritten.

Among the hashtags we selected, we found that "crypto" and "thanksgiving" were much more popular than "lockdown" and "dogs". Our thinking for the popularity of "crypto" is that for finance related topics, there tends to be a lot of bot accounts tweeting and advertising fairly often. This may also be an interesting factor to consider when comparing the texts that come from different hashtags since bots may tweet using significantly different styles than humans. The popularity of "thanksgiving" is obvious since the tweets were collected the week leading up to the holiday. On the day of Thanksgiving, the listener was ran for about an hour and over 30,000 tweets were collected. The remaining two hashtags were not as popular and more susceptible to filtering so our dataset is imbalanced. The table below shows the number of raw tweets we collected, the number of original tweets (after removing retweets, quoted tweets, and non-English tweets), and the number of filtered tweets (after removing tweets with more than one of our hashtags).

Hashtag	Number of Raw Tweets	Number of Original Tweets	Number of Filtered Tweets
#crypto	53061	11857	11838
#thanksgiving	71268	27187	27042
#lockdown	18923	3507	3383
#dogs	19777	2615	2483
Total	163029	45166	44746

Data Cleaning

After collecting all the tweets we began working on cleaning them, starting with removing unwanted tweets that could confuse our models. This entailed removing any tweets that were retweets, quoted tweets, and non-English. The reason for removing the first two is that they contain the same underlying text. In the case of retweets, we did not want our listener to have been collecting while a tweet that got a lot of retweets was published. This would result in the same underlying text to be contained in our dataset multiple times, thus over-representing that tweet. Similarly for quoted tweets, we do not want duplicate tweets to appear in our dataset, as the same tweet could be quoted multiple times. Lastly, we decided to only keep tweets that were in English since different languages have different characters and vocabulary. Fortunately, this was not too difficult to implement as the JSON object which represents tweets has fields which correspond to each of this so it only took a simple check.

Now that we were left with just original tweets, the next part of cleaning dealt with the actual text of the tweets. However, before writing any code for this part, we first wanted to familiarize ourselves with the tweet text from a programmatic perspective. We found that a tweet we see on Twitter itself is very different from how it looks when we output its text as a string. In other words, the "front end" of a tweet looks clean and formatted while the "back end" (what we are dealing with) can be a bit messy. So as we looked at different tweets, we took note of specific components of a tweet's text which we will need to clean in order to make the input to the model more understandable. The list of these components is:

1. One of the other four hashtags
2. Hashtag to which this tweet belongs to
3. Emojis
4. URLs
5. User mentions
6. Non-alphabetic characters (including numbers)
7. Newline and underscore spacing
8. Whitespace

Before we even had looked at the tweets, we knew that we had to decide how we would deal with tweets which contained more than one of our four hashtags. For example, a tweet that was collected using "#crypto" in the listener but also contains "#thanksgiving" in its text. Ideally, this would not occur for hashtags that are distinct enough in context but doing a check on the tweets of just one of our, we found that this was more common of a case than we had initially thought. Thus, we had decided to remove any tweets which contained more than one of our hashtags.

After checking to see if our other hashtags appear in the tweet, we now had to strip the hashtag to which the tweet belongs since leaving it in the tweet would make it too easy for the model and the task of our project pointless. It was important that we did this before any other cleaning because we did want to keep any occurrences of the hashtag as a word as that is important in predicting what hashtag a tweet belongs to. For example, if a tweet had the word "crypto" in it, as well as "#crypto" in it, we would only remove the latter. In other words "crypto" and "#crypto" serve two different purposes in this problem. The former is treated as any other word in the tweet while the latter is the class it belongs to and should not be considered in the input.

While specific emojis could have some contextual meaning for other NLP tasks, we wanted to remove them since they are not actual text and did not feel they would benefit our models. To remove them, we first had to research how Python represents emojis in general so that we could design a function that would easily remove them. We found out that they are represented as Unicode strings and different types of emojis have different codes for these strings. We were able to find out the codes for some of the

common emojis, such as emoticons (faces), symbols, and flags and then use regex in order to replace them with a blank space. While this removed all the emojis in most tweets, there were still other types of emojis that appeared in some of our tweets that we could not remove as we need the specific Unicode representation for all of them.

Handling URL links was relatively easy to do as any link that appears in a tweet starts with either “https” or “http”. Thus, we used a simple regex expression to check for any token that started with either and replaced the entire token with a blank space.

User mentions appear fairly often and start with the “@” symbol and are followed by a username. Again, while some usernames for popular people, such as “@realDonalTrump” could have some contextual meaning, we wanted to remove all user mentions as each unique username would be treated as a unique word in our vocabulary. So we looked for any token beginning with “@” and removed the entire token.

Most of the remaining cleaning now had to do with removing specific characters, mostly non-alphabetic ones. Since tweets are made by anyone with a Twitter account and not proofread text, these characters can appear anywhere in the tweet text. We are only interested in the words being used themselves so we want to make sure we remove any of these characters that are appearing. As we were working on the different parts of cleaning the tweets, we learned that we had to make sure that we ran this part of the cleaning code last. Otherwise, the “@” would be removed from the username and we would not be able to detect user mentions. Similarly, the “#” symbol would be removed from the hashtag which we need to strip so we would not be able to find “#crypto” in the tweet. So, we created a list of characters and if any character in the tweet was in this list, we removed it from the text.

Two characters that we needed to handle and were deliberately left out of this list are the underscore and newline characters. We initially had included them in this list but realized that if we simply replaced it with a blank space, it would cause the tweet to combine words that shouldn’t be. Instead, we decided to replace these characters with a single white space since that is similar to what they are meant to represent. Both the underscore and newline characters are used to separate words so we wanted to make sure this held up in the final version of our tweets.

With all the cleaning done, our final tweets ended up having a lot of whitespace in between them which we wanted to deal with in order to avoid any issues when inputting it into our models. Conveniently, the “split()” function for strings in python is able to recognize where there is white space and just gather the tokens. So, for the remaining text, we split it to get all the tokens in a list and then rejoined them back together with just a single white space in between them. This removed any of the excess whitespace that was appearing between words.

Finally, we had to add the label back to the tweets to ensure that we know their true output (class). To do this, we added the hashtag, without the "#" symbol, to the beginning of the text of the tweet so that when it comes time to reading a tweet, we know exactly where to find its hashtag.

The image shown below is an example of a tweet's raw text and its cleaned text.

```
Raw text:
 📈 #Ethereum (#ETH $ETH) global volume weighted average is currently $548.849 | 8.18% change over 24hrs. #crypto #bl
ockchain #market #news #coin #trading #cryptocurrency https://t.co/0SmN18YDuu 💰

Cleaned text:
 crypto ethereum eth eth global volume weighted average is currently change over hrs blockchain market news coin trad
ing currency
```

Data Processing

Having all the cleaning functions coded, we loaded the file containing the tweets for each hashtag and ran the text through the functions one at a time. This left us with four separate numpy arrays with each corresponding to a hashtag. To save time during the model building process and to prevent having to clean the data multiple times, we concatenated the arrays together and wrote the data as a one column data frame to a .csv file. To run all of the code to load and clean the tweets, it takes about one hour. However, loading a csv file takes only a few seconds. This was our primary motivation for adding the labels to the start of the text and concatenating the arrays together, since we knew it would be faster to read an already cleaned data file in between coding sessions.

The loaded data frame was then separated into two columns, one which contained the text itself and the other which contained the corresponding true label for that tweet. The two columns were then stored as numpy arrays and all of our data was processed and ready to be used. As mentioned in the introduction, we split the data into a training and test set, using an 80-20 split. To validate our models, this training set was split into a smaller training and validation set, using another 80-20 split.

Model Building

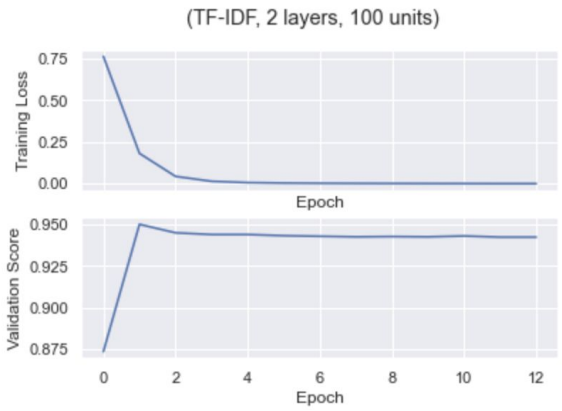
With all of the data collected, cleaned, and processed, our task was to now build various models and see how they perform with our dataset. Vikrant designed the two baseline models we used, which include logistic regression and naive Bayes, while Rahul designed the neural network models with different embedding inputs and architectures.

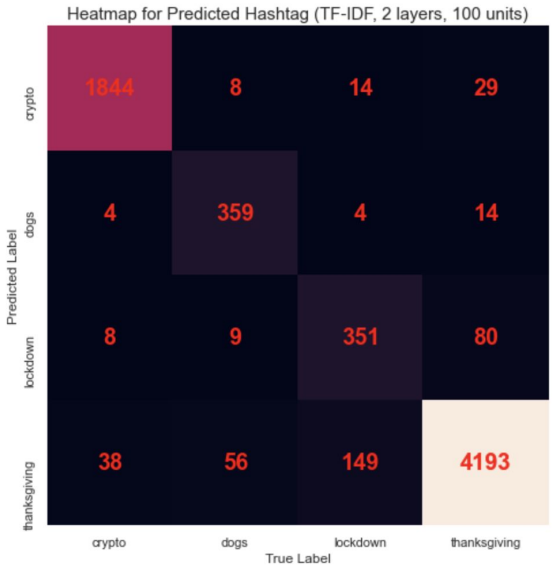
Logistic regression is the baseline supervised machine learning algorithm for classification and it also has a close relationship with the neural networks as neural networks can be seen as a series of logistic regression classifiers stacked on top of each other. We used logistic regression for multinomial classification of our tweets using training data to train the model and validated it over the validation data set. Scikit learn package was used with newton-cg as a solver algorithm to solve the optimization problem

for multinomial classification as this algorithm handles multinomial loss. It has performed pretty well and F1 scores (macro and micro) are much closer to some of our neural networks model.

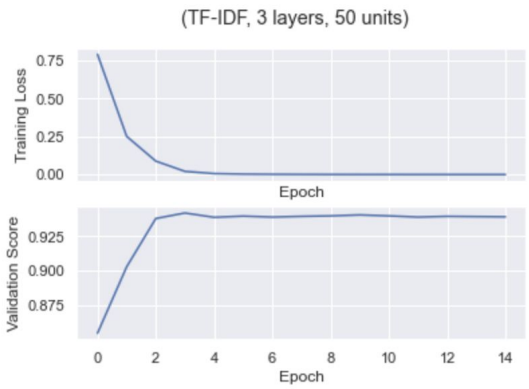
Multinomial naive Bayes classifier is a Bayesian classifier that makes a simplifying (naive) assumption about how the features interact and uses the frequency of each word ignoring the position of the words itself. The “MutinomialNB” method from the scikit learn package was used to train the model based on training data set and was evaluated on the validation set based on F1 measures (macro and micro). It performed poorer than the other baseline model logistic regression.

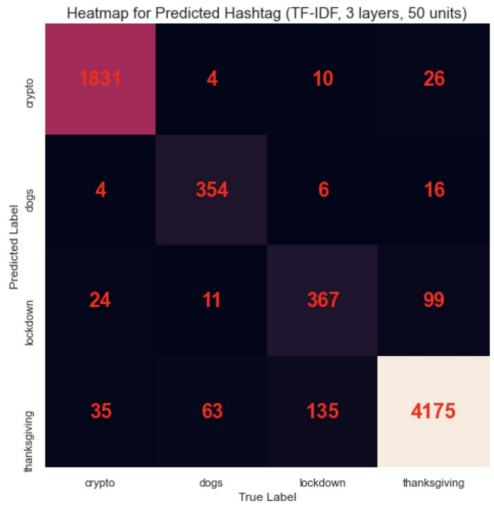
The first set of neural networks that were built involved taking a TF-IDF representation of the tweets. Of these, the first model had 2 hidden layers with 100 units each and used the rectified linear (ReLU) activation function. Initially, this model had only run for about 5 to 6 epochs before it would stop due to the loss or score not changing enough in between epochs. We wanted to see if we could improve the results by allowing the model to train longer so we changed some of the parameters such as the learning rate, tolerance, and batch size. Doing so allowed the model to nearly double the number of epochs it trained for. From the graph below, we can see that the loss starts with a sharp drop and then slowly decreases after about the third epoch. Strangely, we also see the validation score peak at the first epoch and then slowly decrease after. While we are decreasing the bias, this could be a sign that we are starting to overfit the model since the validation score is dropping. Looking at our heatmap, we see that the model does a decent job overall but struggles a bit when it comes to deciphering between the hashtags lockdown and thanksgiving. This was somewhat expected since a lot of people were talking about how they would have to be on lockdown for thanksgiving as they have been advised to avoid traveling for the holiday.





Moving on to the second neural network model we designed, we again used the ReLU activation function, but with 3 hidden layers and 50 units each. Similar to the first model, the training would stop after a few epochs so we changed the parameters and allowed the model to train for a bit longer to see if it would improve the results. Again, we see that the loss drops pretty significantly at the start and then slowly falls off as it gets very close to 0. However, the validation score behaves slightly differently for this model than the first model. It does not appear to have that same peak and then drop off that the first model does. Instead, it gets fairly high and then stagnates. So, if we want to reduce the bias and not overfit as much, model 2 may be better than model 1.

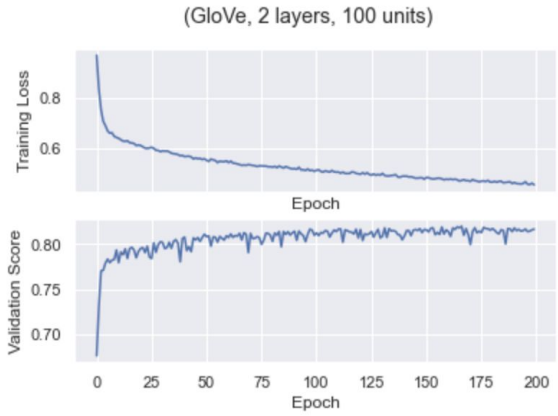


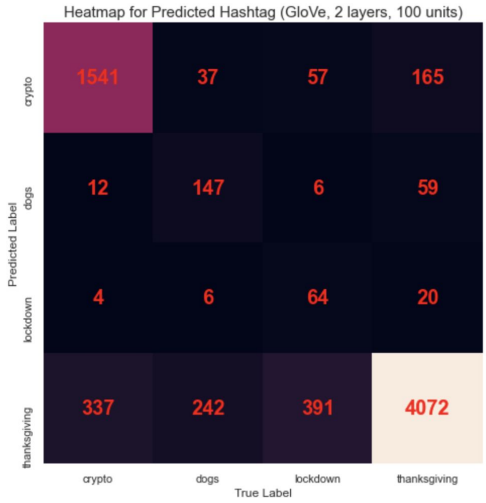


The next set of neural network models we built had the same structure in terms of number of layers and units as the first two models, but instead of using TF-IDF, we used the Twitter GloVe embeddings. We went with the Twitter version instead of the standard embedding since we are working with tweets. We checked the vocabulary of the standard embedding and found that words like “cryptocurrency”, “dogecoin”, and “litecoin” were not in it. However, in the Twitter version they are, which will be useful for the tweets we collected with the “crypto” hashtag. However, even still there are some words like “ethereum” that do not appear in the Twitter embeddings. Also, since we left other hashtags in the tweets, and hashtags are sometimes multiple words without a space in between them, there may be recurring out-of-vocabulary (OOV) words appearing that could be specific to a hashtag. So, in order to handle these cases, we generated a random embedding for those words and saved them in case that same OOV word appeared again.

We noticed that using embeddings allowed the model to train much faster than TF-IDF and reach a greater number of epochs, around 30, before stopping. Since they were able to train much faster, we again wanted to train them longer to see how that would affect our results. For the model with 2 layers and 100 units, we see that the loss actually starts off much higher compared to the TF-IDF models. It then

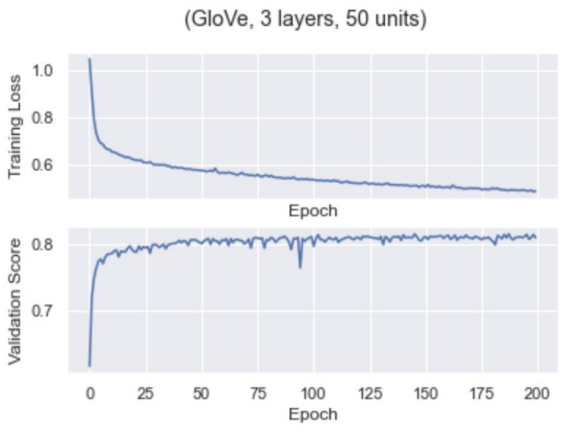
drops a bit and then slowly and consistently gets smaller but only ever gets to about 0.3 whereas the TF-IDF got very close to 0. The validation score shot up and then slowly increased but then stayed fairly constant for the last half of the training.

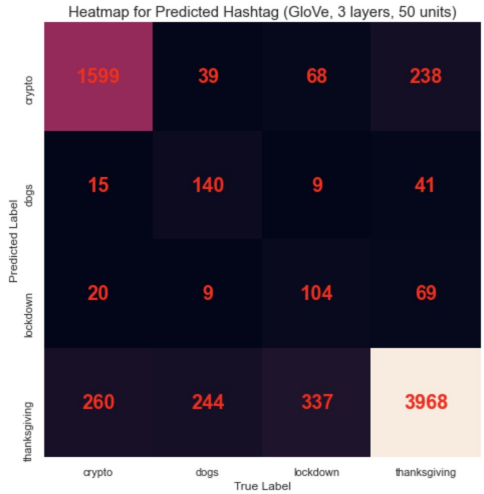




We also replicated the model with 3 hidden layers and 50 units using embeddings and found similar results. The loss sharply dropped then slowly approached 0 but only got to about 0.4 for how long it was trained. The validation score shot up again and went stagnant after about a fourth of the way into the training. So, using embeddings as our inputs to the models showed us that we can continually decrease our loss as we train more but it may not actually do much in terms of improving the accuracy of our model.

Looking at the heatmap, we do see it not perform as well as the TF-IDF models. In particular, we see a lot of the tweets from all of the hashtags being predicted as thanksgiving which is quite strange. For the dogs and lockdown specifically, more of the tweets were getting incorrectly labeled as thanksgiving than their true labels. This may be due to us not having as many tweets from those two hashtags so perhaps collecting more tweets could help out the performance.

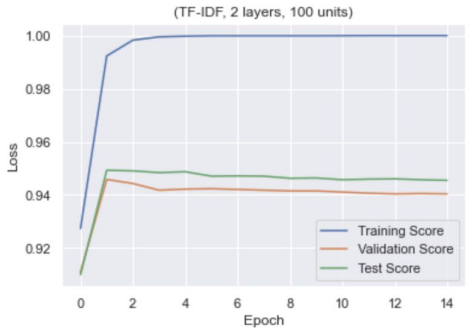




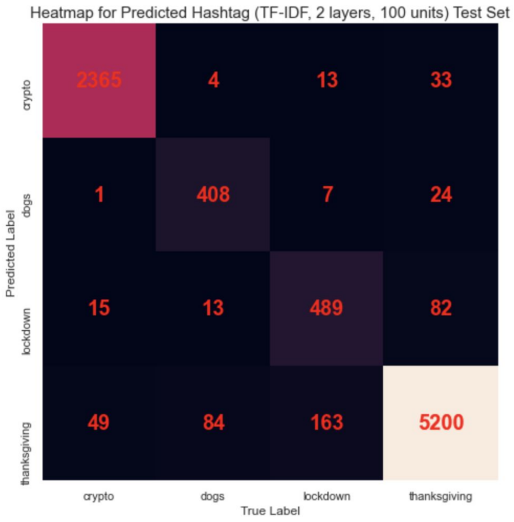
Final Evaluation

Model	Micro F1 Scores	Macro F1 Scores
Logistic	0.932682	0.859822
Naive Bayes	0.870670	0.629563
TF-IDF, 2 layers, 100 units	0.942318	0.885345
TF-IDF, 3 layers, 50 units	0.939525	0.880502
GloVe, 2 layers, 100 units	0.813408	0.590480
GloVe, 3 layers, 50 units	0.811592	0.607849

After building and tuning our models, we compared their macro and micro F1 scores and found the TF-IDF model with 2 hidden layers and 100 units each to be the best so our final step was to see how it did with our test set. From the graphs, we do see that there is overfitting in the model however we were unable to find a method in "sklearn" to deal with this. We do see that the test score is higher than the validation score. In addition, the heatmap for the test set looks much better compared to the validation set. The number of true labeling has increased significantly while the number of errors has only increased by a slight margin.




```
Test Set Micro F1 Score: 0.9454748603351956
Test Set Macro F1 Score: 0.8913604631517256
```



Conclusions

Evaluating each of our models on the validation tests, we found that the neural network models which used the GloVe embeddings performed the worst. This was surprising because we thought that a neural network model would be extremely powerful in prediction, especially when accompanied by a Twitter specific embedding. However, this embedding is likely the reason for its lack of accuracy. The dimension in the embeddings is only 100, compared to the TF-IDF matrix which has dimensions equal to the number of tweets and size of the vocabulary of all the tweets, which is likely close to some number in the hundreds of thousands. So, we can see that the embeddings are a much smaller representation whereas the TF-IDF matrix is a sparse representation of the tweets. This also explains why the models which used the embeddings trained much faster. Thus, it seems like there is a tradeoff between time and accuracy

when we compare using embeddings and a TF-IDF matrix. However, with that, we still believe the TF-IDF models are the best, since even with over 44,000 tweets, these models did not take a substantial amount of time to train (only a few minutes).

Across all the models, it seems to be the case that the most misclassifications had to do with the models predicting a tweet as having the "thanksgiving" hashtag when it actually did not. This leads us to wonder if trendy hashtags, like "thanksgiving", that only are really popular for a short period of time cause a lot of overlap in context. We believe that it is possible people are using the hashtag "thanksgiving" in tweets which may not have a lot to do with the holiday itself, but instead because the holiday is coming up soon.

While we only really worked with the text component of tweets, we did see that tweet data has a lot of fields and aspects which could make for some interesting analyses. It was also fascinating and enjoyable to work with a dataset that we collected and actually use everyday. In addition, this project showed us that NLP tasks can require a lot of cleaning and processing of the data. It reinforced the idea that the more effort we put into handling the data, the better results we expect. So, although modern data and NLP tasks that we are interested in are more complex, if we can put in the extra effort, domain knowledge, and fine-tuning to simplify it, we can improve the results of these models.

Possible Improvements

While our best model did perform fairly well, there are several improvements that could be made as a whole to this project. To start, we could simply pick more distinct hashtags. We noticed that a decent number of tweets were removed due to having overlapping hashtags. Although they were removed, it could be the case that some tweets were not removed but still hold a similar context for both of those hashtags, confusing the model.

In addition to picking more distinct hashtags, it may be useful to pick more popular hashtags than "lockdown" and "dogs" in order to get a more balanced dataset. We saw that a lot of misclassifications occurred when the models would predict a tweet as "thanksgiving", which may be due to the large number of tweets with this hashtag.

In terms of cleaning, there were still some emojis and special characters that we were unable to handle and find an easy way to remove them. Also, we decided to remove numbers since we did not think of them as words. However, for hashtags like "crypto", numbers may actually serve to be very useful in classifying a tweet as such.

Although we modified the parameters of our neural network models, it may be useful to test our other types of models, such as RNNs, bi-directional LSTMs, etc., and see how they perform with our dataset. In addition, incorporating an overfitting reduction technique such as dropout may allow us to find

the model that does not overfit as much. Obviously, this would decrease our training error, but it would highlight the model that does the best on the validation set, which we assume would also perform the best on the test set, which is ultimately the most important measure of a model's performance.

Finally, while not necessarily an improvement to the way we carried out our project, it may be interesting to build models which work with any language and not just English. The difficulty in this is finding a model that can distinguish between the various characters that different languages use, while also understanding the meaning of the tweets to classify them correctly as well.

Summary of Related Research Paper

Title: Comparing automated text classification methods

This paper compares the performances of various machine learning algorithms and lexicon-based approaches on text classification over 41 social media data sets of various sample sizes and languages. The author has found that random forest has performed consistently well for three class sentiments and naive Bayes has performed well on small samples sizes. All lexicon-based approaches performed poorly while support vector machines also performed lower than other machine meaning algorithms. Hence, it is suggested that random forest and naive Bayes are appropriate methods for market research based on social media data.

Marketing researchers are very much interested in classifying large volumes of texts on social media. This text facilitates communication between different consumers, as well as between consumers and firms. By tracking and understanding the continuous and shifting sentiments, they are able to get marketing relevant information and hence, take appropriate actions following that.

Automated classification of unstructured text data into sentiment classes or practically relevant categories has become very important in marketing research to understand how communication contents appear to and affects human readers using social media. Interpretability of these results and the implementational cost have also been of concern while choosing the methods for this classification. The author has compared different methods on these parameters as well, and concluded that naive Bayes and random forest methods do not just perform higher than other methods in terms of classification, but also in their implementation costs. These costs are lower due to easy parallelization of individual decision trees in random forest and the naive assumption of independent features in naive Bayes. In addition, both of these methods have high interpretability. For random forest, there are a few core parameters, intuitive individual trees, and feature importance. For naive Bayes, there are a low number of parameters, as well as an accessible interpretation of conditional probabilities.

References

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.