Tracking Changes in #MeToo Movement with Deep Learning

Shilpa Kancharla
skancha@ncsu.edu

Haitao Jiang
hjiang24@ncsu.edu

I. Motivation

Social media provides a unique opportunity to shed light on phenomena that have previously been ignored or swept under the rug. The #MeToo movement began in 2006 when Tarana Burke first coined the phrase, but a viral Twitter post in 2017 served to lift the hashtag into the mainstream accompanied by the public reckoning of several high-profile men. This hashtag allows people to seek solidarity on a much larger scale in calling out the perpetrators of their abuse. We are going to categorize tweets with #MeToo into the years they were posted. In particular, we look at the years 2017, 2018, and 2019.

After compiling a classification model, we plan to use the `LIME` package to explain the pattern learned by the deep learning model. Such a process allows us to have a direct understanding of the difference the posts of the three years. Specifically, `LIME` provides us with a linear coefficient to every word that appears in one tweet, and tells us which one of them contributes most in categorizing them to one of three years. In using this approach, we can learn the difference by ourselves and explain them to others in language, which is far more better than a black box machine learning model. We list several observations that have a high possibility in each year, and show why our model categorizes it into such category. Some initial discussions are raised at the end of this report.

II. Dataset Description

We use Twitter to find tweets relating to the #MeToo movement. In particular, we use sets of tweets ranging from October 2017 to October 2019. We have obtained these datasets from Kaggle and data.world. In particular, we have one dataset consisting of 350,000 tweets from October 2017, 390,000 tweets from November 2017 to December 2017, approximately 10,000 tweets from October 2018 to December 2018, 600,000 tweets from September 2018 to February 2019, and 15,000 tweets from October 2019. All of these datasets consist of the date the tweet was created and the actual tweet itself, as well as several other columns we eventually discard.

III. Methodology

A. Preprocessing

We clean and merge the four datasets. From each of the raw datasets, we want to get the year a tweet was created and the actual text content of the tweet. We create a dataframe of each of these datasets and drop the unnecessary columns. We convert the date columns of each dataframe to show just the year. After modifying the data column to only display the year, we finally merge all the dataframes together to show the text content of a tweet and the year it was created. We apply further preprocessing to the merged dataframe by first converting all the text to be lowercase. At this point, the number of data points in our dataset is 1,606,028. We then drop duplicate tweets from the dataset which could otherwise lead to bias. This leads our dataset to have 1,048,258 data points. After this, we use regex functions to remove retweets (the 'RT' that appears on retweets), hyperlinks, emojis, mentions (the '@' when mentioning another user), and trailing whitespaces. Our next step in preprocessing is the lemmatization of parts of speech. Lemmatizing a part of speech means that we classify each word as an adjective, adverb, noun, or verb [6]. Each word in the sentence is treated as a token and a tag is given based on the lexical database WordNet [7]. Tuples of tokens and Wordnet tags are then created. If there is a match present, the word is classified (lemmatized) as one of the parts of speech.

Once lemmatization has been achieved, we remove stopwords and words shorter than 3 characters, as such parts of tweets would not have a meaningful impact during analysis. Stopwords are words that are filtered out before or after processing natural language data. They do not add much meaning to a sentence and can safely be ignored without compromising the meaning of the sentence. Upon removing parts of the tweet that will not have a great impact on the meaning of the sentence, we create a new column in our dataframe that holds the text of the tweet with the stopwords and words that are less than 3 characters removed.

B. Exploratory Data Analysis

Before creating our model, we thought it may be interesting to perform some preliminary data analysis. We look at the n-grams of the tweets with no stopwords. An n-gram is a contiguous sequence of n items from a given sample of text or speech. They capture the language structure from a statistical point of view, and give us insight into what letter or word may come after another one [8]. Bar plots for the frequency and n-gram are included in the Appendix section. Respectively, Figures 5 and 6 show the 2 and 3-grams for 2017, Figures (7) and (8) show the 2 and 3-grams for 2018, and finally Figures 9 and 10 show the 2 and 3-grams for 2019. We refer back to these bar plots when we analyze the results from the `LIME` explainer in the Results section.

C. Model Training

We start by using the `train_test_split` function to create `X_train`, `y_train`, `X_test`, and `y_test`. From `keras.preprocessing.text`, we use the `Tokenizer` function to create a token object to vectorize a text corpus, by turning each text into either a sequence of integers. We then used the `fit_on_texts` function to update the internal vocabulary based on a list of texts, which `X_train` consisting of the tweets with no stop words. Upon completing this step, we apply the `texts_to_sequences` function on `X_train` and then `X_test`, thus creating the `X_train_seq` and `X_test_seq` input data, respectively. This last function transforms each text in `X_train` and `X_test` to a sequence of integers. We then use the `pad_sequences` function to transform `X_train` and `X_test` into a two-dimensional `numpy` array of `(number of samples, number of timesteps)`. The number of timesteps is equivalent to the maximum length of a sequence, which is `X_train_seq`. Simply put, this is the length of the longest sequence in the list. We set the padding to be post to pad after each sequence. As a final preprocessing step, we apply one-hot encoding to `y_train` and `y_test` through the use of `LabelBinarizer` from `sklearn.preprocessing`.

TABLE I
STRUCTURE OF ONE-DIMENSIONAL CONVOLUTIONAL NEURAL NETWORK

Layer (type)	Output Shape	Parameter No.
Embedding	(None, 74, 200)	74911400
Conv1D	(None, 72, 50)	30050
GlobalMaxPooling1D	(None, 50)	0
Dropout	(None, 50)	0
Dense	(None, 3)	153

The CNN takes input as a tokenized sequence of words. We apply `Embedding`, `Conv1D`, `GlobalMaxPooling1D`, and Dense layers to define the CNN model. A summary of this structure is shown in Table 1 above. The `Embedding` layer turns positive integers into a fixed length vector of defined size. The resulting vector is a dense vector that has real values, as opposed to zeros and ones. Embedding layers work like a lookup table. The words are keys in this table, while the dense word vectors are akin to values. The `Embedding` takes in three parameters: `input_dim` (size of the vocabulary), `output_dim` (length of the vector for each word), and `input_length` (maximum length of a sequence).

In addition to creating a CNN model, we also created a FastText and Long Short-Term Memory (LSTM) model. The structure of each model is shown in Table 2 and 3, respectively. In the FastText model [1], every word is represented as a bag of character n-grams. After associating a unique vector to each n-gram, sentences are represented as the sum of the vector representations. The class of each sentence is predicted by a multilayer softmax function based on the Huffman tree algorithm. We can also apply LSTMs to text classification problems because they can learn long-term dependencies; such models can be used to perform classification on sequential data. Moreover, an LSTM cell does not treat every data point (in this case, a data point is a word) as an uncorrelated sample [2].

TABLE II
STRUCTURE OF FASTTEXT NEURAL NETWORK

Layer (type)	Output Shape	Parameter No.
Embedding	(None, 83, 3)	1123671
SpatialDropout	(None, 83, 3)	0
GlobalMaxPooling1D	(None, 3)	0
Dropout	(None, 3)	0
Dense	(None, 3)	12

TABLE III
STRUCTURE OF LONG SHORT-TERM MEMORY NEURAL NETWORK

Layer (type)	Output Shape	Parameter No.
Embedding	(None, 83, 3)	1123671
LSTM	(None, 125)	64500
Dropout	(None, 125)	0
Dense	(None, 3)	378

D. Model Selection

After running our preprocessed training and validation data through the 1D-CNN, FastText, and LSTM models over 10 epochs, we achieved the following accuracies on the preprocessed test dataset shown in Table 4 as a baseline (with a dropout value of 0.5 where it applies).

TABLE IV
ACCURACIES FOR BASELINE MODELS

Model	Accuracy
1D-CNN	82%
FastText	75%
LSTM	61%

Initially, we decided to perform hyperparameter tuning based on the dropout rates, L1 regularization multiplier, and L2 regularization multiplier. There are 48 different combinations given grids above, and we will use following metric to decide which of them is the best:

$$L(r, l_1, l_2) = \frac{1}{5} \sum_{i=n-5}^{n} F_1(r, l_1, l_2),$$

where the j^{th} of $F_1()$ function is the j^{th} F1 score calculated from the training set. We are using the mean of the last five f1 scores as the indicator because we have some preliminary experience of a bumpy curve. A final model using these parameters with more (15) epochs will be trained using all training data, then we can predict results for the testing files using our final model. Our initial values for dropout was 0.1, the L1 regularization value was 2^{-6}, and the L2 regularization value was 2^{-8}. We produced a classification report from predicting the values of the `X_test_seq` as the input data. Our initial classification report yields the following information in Table 5:

TABLE V
CLASSIFICATION REPORT FOR 1D-CNN WITH L1 AND L2 REGULARIZATION

Label	Precision	Recall	F1-Score
2017	0.55	0.65	0.60
2018	0.68	0.69	0.68
2019	0.62	0.24	0.35

From this model, we achieved an accuracy of 82% overall. In order to handle any issues with class imbalance, we used the `class_weight` function in the `sklearn` package. We were also curious to see what the classification results for our model would be if we were to only use dropout as the only regularization method, and not use L1 or L2 regularization. Upon attempting to use several different dropout values, we came to use 0.3 as our final dropout value for our CNN model. The classification report of using this model yields the following in Table 6:

TABLE VI
CLASSIFICATION REPORT FOR 1D-CNN WITHOUT L1 AND L2 REGULARIZATION

Label	Precision	Recall	F1-Score
2017	0.87	0.92	0.90
2018	0.84	0.92	0.70
2019	0.66	0.24	0.35

We achieved an overall accuracy of 85% with this model that only uses dropout regularization and choose this as our classification model. Furthermore, we note that this model does not use the `class_weight` function in the `sklearn` package. We attempted to use the `class_weight` function and yielded an accuracy of 75% over the course of 10 epochs. Further results of this final model are discussed in the following section.

When tuning the Fast text and LSTM models, the dropout values is the hyperparameters that we are interested in. We do cross-validation on three possible values $rate = [0.1, 0.4, 0.7]$ and find out that the optimal model can be achieved by selecting $rate = 0.4$ in both scenarios. The fast text reaches a accuracy of 0.83 and LSTM gets 0.61. We can see from the classification report that they all surfer from the precision of 0 in the 2019's tweets, which might due to the fact that we have a smaller size of 2019 training data, compared with other years.

IV. RESULTS

A. Performance and Final Model

The loss and accuracy plots of the CNN with dropout regularization are shown in Figure 1 and **??** below, respectively. A validation dataset was created from the `X_train` and `y_train` partitions. We split the training and validation sets to 70% and 30% of the original training set in that order. As evinced by Figure 1 and **??**, the validation dataset's loss and accuracy values experienced little to no change over the course of 10 epochs. Moreover, we believe that this may indicate against overfitting in our current model as there is not a huge difference between the training validation loss and accuracies over the course of 10 epochs.

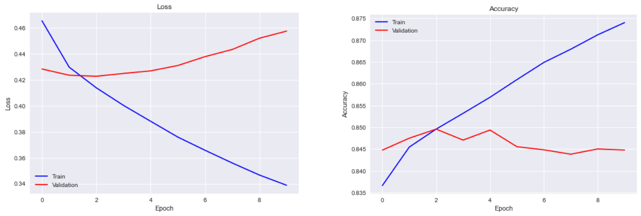

Fig. 1. Loss and accuracy on training and validation sets

Moreover, we believe the low recall, precision, and F1-score values shown in the classification reports of Table 5 and Table 6 for 2019 have resulted from the lack of 2019 data in comparison to 2017 and 2018 data. Specifically, the amount of tweets from each year can be shown in Table 7.

TABLE VII
CLASSIFICATION REPORT FOR 1D-CNN WITH L1 AND L2 REGULARIZATION

Year	Number of Tweets
2017	286269
2018	638897
2019	116199

As the level of imbalance increases, the accuracy is bound to increase [4]. Though our accuracy level of 85% may be acceptable or judged as decent, the cost of misclassifying 2019 data entries plays a factor in achieving this accuracy. Even if we to misclassify all 2019 entries and hypothetically identify all the 2018 and 2019 tweets correctly, we would still achieve an accuracy of nearly 89%. We believe that if we were handle the data imbalance more appropriately, it could be further possible to improve this model. We expand upon this idea in the next section.

B. Interpreting Classification Results

`LIME` is able to explain the results of black box classifiers such as our CNN model with two or more classes. We compare the findings from the explainer from `LIME` to our finds from performing exploratory analysis with 2 and 3-grams, figures for which have been given in the Appendix. To explain the `LIME` visualizations in the following figures of this section, we see that our model predicts the the texts provided should be classified as either 2017, 2018, or 2019 with certain probability. The words shown in the bar plot are shown with their weights of how much they contribute to a tweet belonging in a certain category. We start by examining some of the `LIME` Explainer results for the year 2017. Such an example is shown in Figure 2 below.

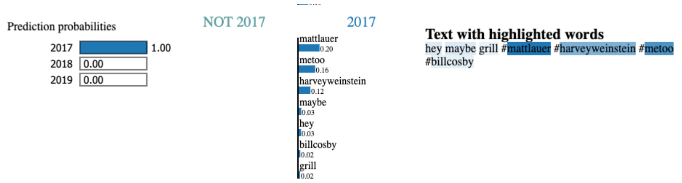

Fig. 2. 2017 LIME Explainer Result

We observe that in comparison to the 2 or 3-gram bar plots depicted in Figures 5 and 6 that a 2-gram like ’bill cosby’ or ’matt lauer’ are not captured, but they contribute to the probability of a tweet being classified as being from 2017 to some extent. The visualization shown in Figure 2 demonstrates that our model would classify this tweet as 2017 with a probability of 100%. If we were to remove ’mattlauer’ from the tweet, we would expect the model to predict the 2017 label with a probability of 100% - 20% = 80%. We also take a look at the LIME Explainer results for the year 2018 in Figure 3.

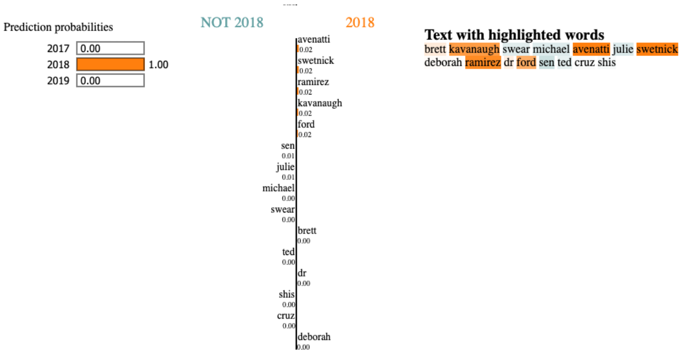

Fig. 3. 2018 LIME Explainer Result

While the 3-gram bar plot for 2018 in Figure 3 shows that ’christine blasey ford’ is a common 3-gram for that year, our initial analysis did not pick up ’brett kavanaugh’ as a 2-gram for 2018, although these two names are related due to the Kavanaugh Trial from 2018. Next, we examine the LIME Explainer results for the year 2019 in Figure 4.

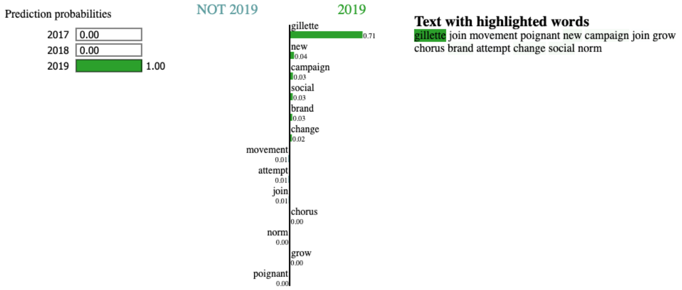

Fig. 4. 2019 LIME Explainer Result

In the various LIME Explainer results for 2019, the word ’gillette’ came up repeatedly. This is in agreeance to Figure 10 in the Appendix, as ’gillette ad’ comes up as a frequent 2-gram. Although, the frequency of this 2-gram is lower than other 2-grams shown in Figure 9.

We believe that using LIME to visualize the results of our 1D-CNN is evidence that in addition to plotting the n-grams from the exploratory data analysis, our model is able to discover more words in tweets that may contribute it belonging to a certain year. At minimum, the results displayed from the LIME visualizations further corroborate our findings in the n-gram bar plots. We believe that our 1D-CNN model and further improvements to it could better classify tweets into their respective years by finding patterns in text data that are not picked up by finding the most common n-grams.

V. Further Directions

In order to deal with the data imbalance displayed in Table 4, we propose using data augmentation techniques such as a random oversampling technique like SMOTE [5]. We hypothesize that having more minority class data points that are synthetically generated could counter for the lack of 2019 data points. An alternative to using data augmentation would be to mine Twitter for #MeToo tweets from the year 2019. We hypothesize that if we account for this data imbalance and attempt to apply our CNN model, we may see better precision, recall, and F1-score values for the 2019 class specifically. In addition, we believe that this project can be extended to examine tweets related to #MeToo from 2020 and 2021, to see what topics are relevant to more recent years as well.

Moreover, we would like to revisit the idea of using an LSTM in text classification without L1 and L2 regularization. We suspect that adding terms of L1 and L2 regularization may depreciate the performance of RNNs, we found that L1/L2 regularizing the RNN cells also compromises the cells’ ability to learn and retain information through time. While using L1 and/or L2 penalties, the recurrent weights can help with exploding gradients, this approach can also limit the model to a single point attractor at the origin, where any information inserted into the model dies out exponentially fast. This prevents the model from learning efficiently [3].

References

[1] Bojanowski, P., Grave, E., Joulin, A., Mikolov, T. (2017). *Enriching word vectors with subword information* (Transactions of the Association for Computational Linguistics), 5, 135-146.

[2] Rao, A., Spasojevic, N. (2016). *Actionable and Political Text Classification using Word Embeddings and LSTM* (arXiv.org).

[3] Pascanu, R., Mikolov T., Bengio Y. (2013). *On the difficulty of training recurrent neural networks* (arXiv.org).

[4] Somasundaram, A., Reddy U. (2016). *Data Imbalance: Effects and Solutions for Classification of Large and Highly Imbalanced Data* (ResearchGate).

[5] Mishra, S. (2017). *Handling Imbalanced Data: SMOTE vs. Random Undersampling* (IRJET).

[6] Agarwal, V. (2015). *Research on Data Preprocessing and Categorization Technique for Smartphone Review Analysis* (International Journal of Computer Applications).

[7] Fellbaum, C. (2010). *WordNet* (Theory and Applications of Ontology: Computer Applications).

[8] Violos, J., Tserpes, K., Varlamis, I., Varvarigou, T. (2018). *Text Classification Using the N-Gram Graph Representation Model Over High Frequency Data Streams* (Theory and Applications of Ontology: Computer Applications).

VI. Appendix

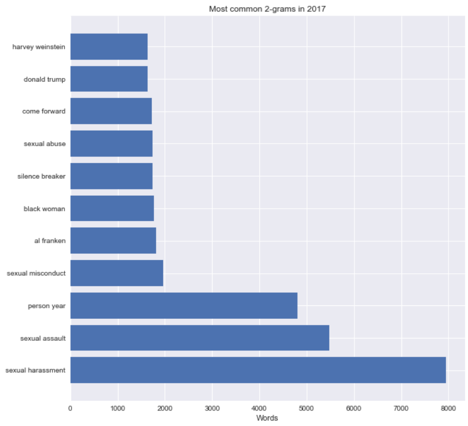

Fig. 5. Most Common 2-grams in 2017

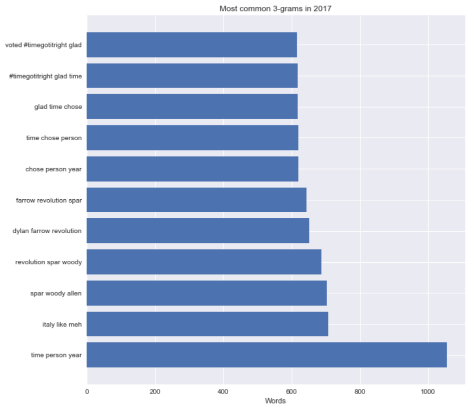

Fig. 6. Most Common 3-grams in 2017

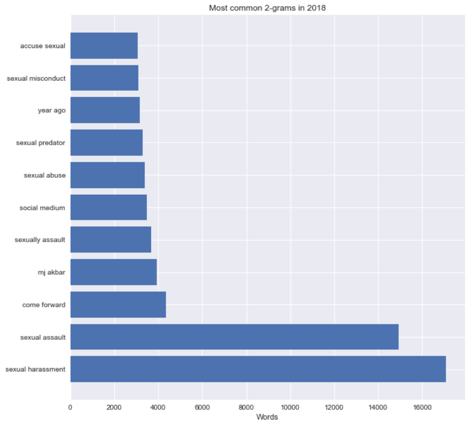

Fig. 7. Most Common 2-grams in 2018

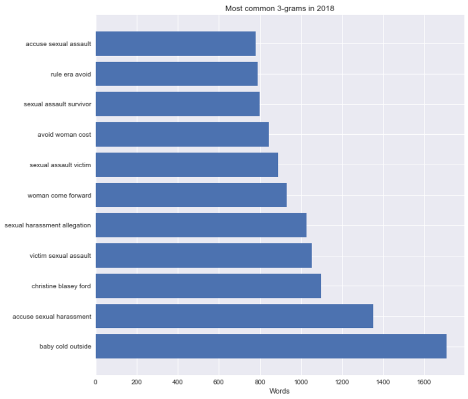

Fig. 8. Most Common 3-grams in 2018

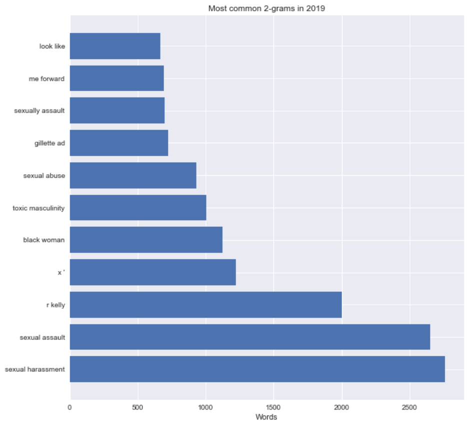

Fig. 9. Most Common 2-grams in 2019

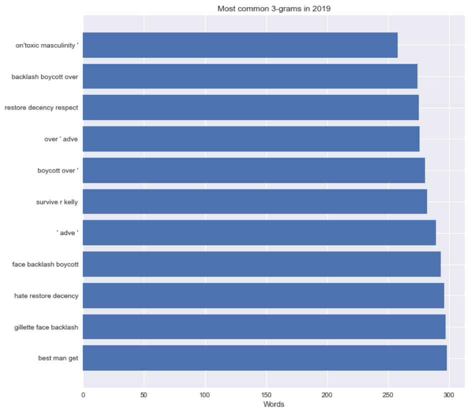

Fig. 10. Most Common 3-grams in 2019