CUNY School of Professional Studies Master of Science in Data Science Capstone Project

Identifying School Shooters in their Digital Writings

Joseph Connolly

jmconnolly95@gmail.com

Advisor:

Professor Paul Bailo PhD

CUNY School of Professional Studies

Fall 2022

Table of Contents

Abstract

At the time of this writing, the United States of America has seen approximately 352 school shootings since the Columbine High School shooting on April 20th, 1999. According to the Washington Post, this includes more than 311,000 students who have been exposed to gun violence at a K-12 school. * The Federal Bureau of Investigation (FBI), the U.S Secret Service (USSS), and psychologist- school shooter expert Dr. Peter Langman, all claim they do not decide to carry out their attack in an instant, but rather over a long period of time. There are many factors that come into play that make a school shooter, such as upbringing, domestic life, socialization within school, as well as their personality and psychological attributes. For school shooters, these attributes generally result in anti-social, paranoid, revengeful, hateful, depressed, narcissistic, and schizotypal features that are all well documented, observed, and recognized among past school shooters. Leading up to their attack, these shooters have written in personal journals, created stories, autobiographies, blog posts, and/or made videos of themselves using words, language (in a semantic sense), and speech that are all identifiable when put into the format of a written document from those who are not school shooters.

Keywords

Violence Detection, Gun Violence, Leakage, Machine Learning, Natural Language Processing, Preventative Intervention, Python, School Shooters, USA, Identification, Text Analysis

Introduction

Leading up to an attack, many school shooters exhibit "leakage", which is a major indication of their intention to carry out an attack, or cause harm or death to others. Webster's dictionary defines leakage as "becoming known despite efforts at concealment". In the context of school shooters, this is an expression of a true motivation to attack which can be expressed by online comments, video recordings/vlogs, diary/journal/blog entries, messages, postings on forums, literary works, art, and so on. Witnessing leakage provides an opportunity to anticipate and potentially stop their heinous acts. A recent example of leakage can be found in the case of Salvador Ramos, the school shooter in Uvalde, Texas, who was not prevented from his school shooting, admitted he was going to "…shoot up [an] elementary school [right now]". While this statement of leakage is particularly obvious especially after he claimed he'd just murdered his grandmother, his preceding interactions online, among friends, as well those who saw him in his community gave clues to what he had in mind and intended.

According to documents published by the Texas House Committee who investigated this incident, these indications of leakage included pictures of his receipts for guns and ammunition, video tutorials on how to load a gun, witnesses describing his presence as "creepy" and "school-shooter like" at the gun store, as well as his admittance that he would be known for something within a certain amount of time. In this analysis, many of the documents involved are comprised exclusively of leakage, such as the manifesto by Seung-Hui Cho. Others are not necessarily, such as Adam Lanza's vlogs about anti-natalism (the idea that having children is wrong) for instance. Using machine learning and Natural Language Processing, this paper aims to build a model to classify a school shooter according to digital writings among those who are not school shooters.

Literature Review

Overview

Despite the seemingly increasing number of annual school shootings that have occurred in the US since 1999, these unfortunate events are in fact quite rare and unlikely to happen. Yet, because of the massive media coverage that's entailed as well as the empathy many of us share for the victims and their families, they leave a mark on the psyche of the American public. Trust in the public safety system dwindles and mental health generally worsens, especially for staff members of schools, students, and parents. Nonetheless, the RAND Corporation reports that school shooter events are rare and only account for 0.5% of all homicides (calculated in 2020 by Duwe). Even in the 2021-2022 schoolyear, where 43 school shootings took place according to data provided by the Washington Post, the approximate probability of any one student being a school shooter at a public PreK-12 school during the 2021-2022 schoolyear is 0.000087%. To give a sense of how rare that is, one has a greater chance of being struck by lightning (0.0065%), but a lesser chance of winning the Mega Millions Jackpot (0.00000039%).

However, thanks to intervention signals and methods published by the FBI and Secret Service, who independently published methods of detection and prevention for school shootings, we presumably do not have a higher number of incidents than what it is now. It is important to note that these do not consider "mass shooter" events. According to the DOJ, a mass shooting event is defined in which four or more people are injured or killed by the gunfire from the perpetrator. Consider the shooting that happened at a Brooklyn subway station in New York City, or the racially motivated one in Buffalo NY (both events that occurred in 2022). However, like school shooters, both perpetrators exhibited similar characteristics of school shooters as well as their own version of leakage. Yet, a mass shooting event can also involve incidents of road-rage or random arguments in which perpetrators seem to "snap" with no clear agenda other than the intention to cause harm and/or death immediately. Therefore, identifying "mass shooters" is more complex and broad than identifying school shooters. Of course, many school shooters are mass shooters, but this paper focuses on school shooters particularly, and does not take into consideration the broader definition of a mass shooter.

Psychology of School Shooters

Dr. Peter Langman is a researcher and psychologist who has studied, counseled, prevented, and dissuaded potential school shooters. He regularly publishes literary data from all types of school shooters from around the world which includes manifestos, journals, video transcriptions, literary works, blog/web/forum posts, court documents, police interviews, and more. A dataset for building this model will be compiled from his website, schoolshooters.info, for training and testing which will feature writings and video transcriptions of school shooters that were completed before or on the day of their attack. In an article for Psychology Today, Dr. Langman claims that there is not a specific profile of a school shooter. He argues they are particularly complex individuals that don't uniformly follow the same pattern in their journey of becoming a school shooter. Though, they typically fall into one of three categories: psychopathic, psychotic, and traumatized. He indicates a distinction between K-12 shooters and College/University shooters, which is that traumatized shooters were the most frequent among shooters at K-12 schools. However, more than 50% of shooters at colleges/universities were immigrants or international students. A majority of the younger shooters came from poverty stricken homes, where they often encountered violence and chaos domestically. The college/university shooters seemed to have "longstanding grievances" with the school and exhibited very glaring signs of leakage (i.e Seung Hui Cho). Yet, the "random" shooters who did not appear to necessarily have as many or any grievances with the school gave off fewer warning signs. Their attacks turned out to be deadlier, with killing and wounding over 5 times the amount of those who planned their attack. Ultimately, Dr. Langman concludes that in addition to being psychopathic, psychotic, or traumatized, they experienced continuous failures in their aspirations within academia, the military, work, romantic/intimate relationships, finances/financial stability, and in their own sense of masculinity.

In a paper by Dr. Yair Neuman entitled “Profiling School Shooters: automatic text-based analysis”, written additionally with Dan Assaf, Yochai Cohen, and James L. Knoll, an analysis was performed that concentrated on detecting features of personality disorders found among school shooters. These included depressive, paranoia, narcissistic, and schizotypal traits. Using DSM-V criteria and Millon’s personality traits, they built a semantic vector to measure the emotion contained in the writing of the texts by the school shooters. This discovery was attributed to a paper done by Neuman and Cohen themselves, entitled “A Vectorial Semantics Approach to Personality Assessment”. This quantified emotions and features of personality disorders expressed in the school shooters’ writings, as well as the degree of the expression of these features. While the methods are beyond the scope of this paper, their findings proved that there are detectable traces of depressive, narcissistic, schizotypal, revengeful, humiliating, lonely, helpless, abandoned, unsafe, chaotic, and paranoid features, as well as the degree to which they were all expressed within writing. This alludes to evidence that there is a detectable trace of school shooters according to their writings.

Preparing Literary Data for Text Analysis

Preparing textual data for any sort of Natural Language Processing (NLP) analysis involves a standard method of preparation. This entails lowercasing all words, removing unwanted characters and punctuation marks, tokenization of the text, removing “stop words”, and lemmatizing or stemming tokenized words. By lower casing all words, the same word repeating in different casings will be recognized as the same word rather than a different word. For instance, even though the words “Dog”, “DOG”, and “dog” are the same word, they will all be recognized as three different words because of the casing of the various characters, unless lowercased. Removing unwanted characters or texts such as URL links or newline indicators should also be taken out since they do not add any value to the textual analysis at hand. Additionally, stop words should be removed since they are words that appear frequently and do not add value to the context of the writing. These include words such as “the”, “and”, “a”, “an”, as well as pronouns. Next, the remaining words in the text are tokenized, which turns the string of text into a list of text and individualizes each and every remaining word with quotes. Finally, the tokens are either lemmatized or stemmed. These methods entail transforming the tokenized words into their root form, such as the words “younger” being reduced to “young”, as well “considering” to “consider”. However, lemmatizing and stemming produce slightly different results. While they both aim to achieve the same thing, stemming appears to boil down words to a root form that is not necessarily intuitive to human readers, whereas lemmatization does. For example, the following are images of tokenized words that were stemmed and lemmatized respectively:

Figure 1: Stemmed Text

```
print(stemmed_tokens[0:21])
```

```
['’', 'understand', 'younger', '’', 'alway', 'immens', 'hatr', 'cultur', 'consid', 'cultur', 'delusion', 'valu', 'human', 'mindlessli', 'coerc', 'onto', 'spread', 'differ', 'disea
s', 'previous', 'sought']
```

Figure 2: Lemmatized Text

```
print(lemmas[0:21])
```

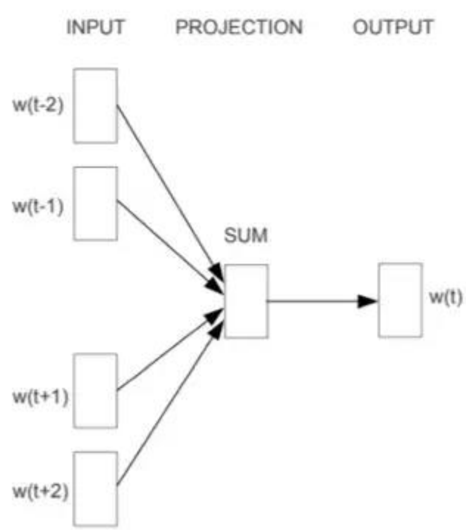
```
["'", 'understand', 'young', "'", 'always', 'immense', 'hatred', 'culture', 'consider', 'culture', 'delusional', 'value', 'human', 'mindlessly', 'coerce', 'onto', 'spread', 'differe
ntly', 'disease', 'previously', 'seek']
```

As we can see, certain words do not make sense in the stemmed method of text. The words “always”, “immense”, and “culture” were stemmed down to “alway”, “immens”, and “cultur”, whereas the lemmatized text maintained the word structure and thus its sensibility to the human reader. While this overall processing of cleaning/dissecting bodies text seems unintuitive since the documents are less sensical, it’s important to do so because computers cannot read nor understand English. To help a computer “understand” text, everything is to be converted into a number. By cleaning texts, we strip away all unnecessary words, tenses, and parts of speech that allow for an understanding of texts by the computer in such a way that we would like it to understand something if we were to read it.

Feature Extractions: Doc2Vec

The features for identifying school shooters’ writings are obtained with the Doc2Vec algorithm that’s available in gensim. While it was implemented in python for this paper, gensim is also available in R as well. Doc2Vec is an unsupervised machine learning algorithm that converts documents to numeric vectors known as Paragraph Vectors (PV). Doc2Vec is also able to maintain the relationship of words upon its conversion. For instance, it’ll dictate a strong association with the words “person”, “man”, and “worker” as they are all personable nouns. It’ll also be able to predict the following word in a sentence, like “the dog drank out of his [bowl]”, where “bowl” can be anticipated by the paragraph vector-distributed bag of words algorithm (PV-DBOW). This predicts the next word according to the context.

Figure 3: Functionality chart of PV-DBOW



PV-DBOW

To illustrate how a paragraph vector is formed, we are given a sequence of words from the training set, where each word is mapped to a unique vector displayed by a column in a matrix, *W*. Each column is indexed by the position of the word in the vocabulary, and the sum of the vectors are used as features for predicting sequential words in a paragraph. The word vector model is to maximize the average log probability of the word, w_t, by the following:

$$\frac{1}{T}\sum_{t=K}^{T-k} log\left(P(w_t|w_{t-k},\ldots,w_{t+k})\right)$$

Equation 1: Maximization of log probability

The prediction is done with a multiclassification function via softmax:

$$p(w_t|w_{t-k},\ldots,w_{t+k}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

Equation 2: Multiclassification Softmax function

Where each y_i is an unnormalized log probability of each input-word, *i,* determined by:

$$y_{\mathrm{i}} = b + Uh(w_{t-k},\ldots,w_{t+k};W)$$

Equation 3: probability of words

Where *U,* and *b* are the softmax parameters, and *h* is built by an average of word vectors pulled from the matrix, *W.*

Next, the Distributed Memory (DM) is established by the neural network-based vectors trained upon a stochastic gradient descent algorithm, where the gradient is obtained by backpropagation. Essentially, this acts as a memory which remembers the missing context or topic of a paragraph. In the analysis, the distributed memory will be utilized as a feature.

In conclusion, the features obtained for this paper were the Paragraph Vector Distributed Bag of Words, and the Distributed Memory model of Paragraph Vectors. In the analysis, results are shown for classifiers with PV-DBOW, and with a combination of PV-DBOW + DM, to which the authors, Quoc Le and Tomas Mikolov both recommended using the combination of. Below is how the PV-DBOW model was instantiated.

Figure 4: Creating vocabulary vector via CBOW in Doc2Vec

```
# Vocabulary

model_dbow = Doc2Vec(dm= 0, vector_size =300,
                     negative = 5, hs = 0,
                     min_count = 2, sample = 0,
                     workers = cores)

model_dbow.build_vocab([x for x in tqdm(train_tagged.values)])
```

```
100%|██████████| 1729/1729 [00:00<?, ?it/s]
```

After instantiating the PV- DBOW model, it is then initialized and trained on a neural network to identify the tagged values within the training set upon 30 epochs. Upon its completion of training, we are left with numeric values.

Figure 5: Initiating and training the PV-DBOW model

```
%%time
for epoch in range(30):
    model_dbow.train(utils.shuffle([x for x in tqdm(train_tagged.values)]),
                     total_examples = len(train_tagged.values),
                     epochs =1)
    model_dbow.alpha -= 0.002
    model_dbow.min_alpha = model_dbow.alpha
```

Now we obtain the distributed memory (DM) feature. Combining this with the PV-DBOW feature improved F1 scores substantially as the results will show. The DM model instantiated similarly as the PV-DBOW model, trained, and is then concatenated with PV-DBOW.

Figure 6: Instantiating and training Distributed Memory model.

```
model_dmm = Doc2Vec(dm=1, dm_mean = 1, vector_size = 300, window = 10, negative  =5,
                    min_count = 1, workers = 5, alpha = 0.065, min_alpha = 0.065)

model_dmm.build_vocab([x for x in tqdm(train_tagged.values)])
```

```
%%time

for epoch in range(30):
    model_dmm.train(utils.shuffle([x for x in tqdm(train_tagged.values)]),
                    total_examples = len(train_tagged.values), epochs = 1)
    model_dmm.alpha -= 0.002
    model_dmm.min_alpha = model_dmm.alpha
```

Figure 7: Concatenating both models for the final feature

```
from gensim.test.test_doc2vec import ConcatenatedDoc2Vec
```

```
new_model = ConcatenatedDoc2Vec([model_dbow, model_dmm])
```

```
def get_vectors(model, tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0],
                                 model.infer_vector(doc.words)) for doc in sents])
    return targets, regressors
```

Data and Analysis

Overview

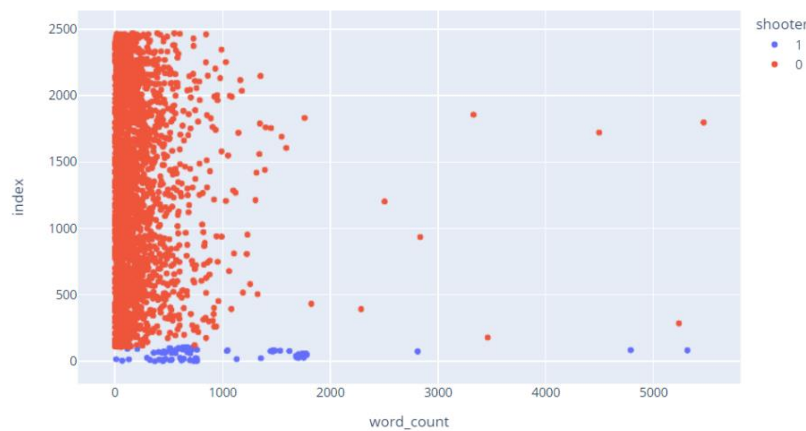
The dataset used for this analysis is comprised of two datasets: various documents and writings by school shooters from schoolshooters.info, and the Blog Authorship Corpus dataset containing the writings of 19,000 bloggers obtained from Kaggle.com. The former was created and saved into a .csv file with four columns: *author*, *topic*, *text*, and *shooter*. The final column, *shooter,* is indicative of whether the author was a school shooter with a 0 for no, and 1 for yes. Due to the rarity of a school shooting taking place, the data for the shooters was not randomly sampled. It was carefully selected out of necessity to choose writings or video transcriptions from the subjects themselves that were created before or on the day of their attack. No court records were included.

Again, because of the infrequency of this type of data, the school shooters were not limited to American individuals or shootings at high schools. This includes shootings at colleges, such as Seun Hui Cho (Virginia Tech, 2004), Marc Lepine (Ecole Polytechnique de Montreal, 1989), Gang Lu (University of Iowa, 1991), and Elliot Rodger (University of California Santa Barbara, 2014). Aside from those mentioned, the ones from other countries include Weillington De Oliveria (Brazil, 2011) and Pekka-Eric Auvinen (Finland, 2007). The remaining shooters include Adam Lanza (Sandy Hook, Connecticut, 2012), Dylan Klebold and Eric Harris (Columbine, Colorado, 1999), Nikolas Cruz (Parkland, Florida, 2018) Kenneth Bartley (Jacksborough, Tennessee 2005), Alvaro Castillo (Hillsborough, North Carolina, 2006), and Robert Butler (Omaha, Nebraska, 2011).

The entries from the bloggers' dataset were selected completely at random with a sample of about 2400 entries out of the 260,000 total entries. In order to combine the two sets, the bloggers' set was reduced to the following columns: *id, topic, text,* where *id* was renamed to *author.* The column *shooter* was added as well to indicate none of the bloggers were school shooters with the value 0. Both sets were merged in R in descending order according to author name. Looking at the distribution of data, we can see a distinct imbalance of the writings of the shooters compared to the bloggers both terms of the length of their writings, as well as the overall word count. The percentage of words for bloggers and shooters works out to 95.7% and 4.3% respectively.

Figure 8: Length of entries by bloggers (purple) and shooters (green)

Figure 9: Word count by bloggers (red) and shooters (blue)



Preparing the Data

The text was cleaned in the following way via cleaning function that was written in python:

1) Lower casing all text
2) Removing the whitespace
3) Removing the punctuations
4) Tokenizing the text
5) Lemmatizing the tokens
6) Removing stop words
7) Part of Speech-Tagging the remaining lemmatized words as nouns, adjectives, or verbs.

This process standardizes the text within the data, eliminates unnecessary words, and makes the dataset more concise as to the meaning and context of what's being written.

Emotional Insights

Thanks to the NRCLex package, it's possible to obtain 10 different emotions of text. Looking at the charts below, we can see the total counts of each emotion detected for the writings by the school shooters.

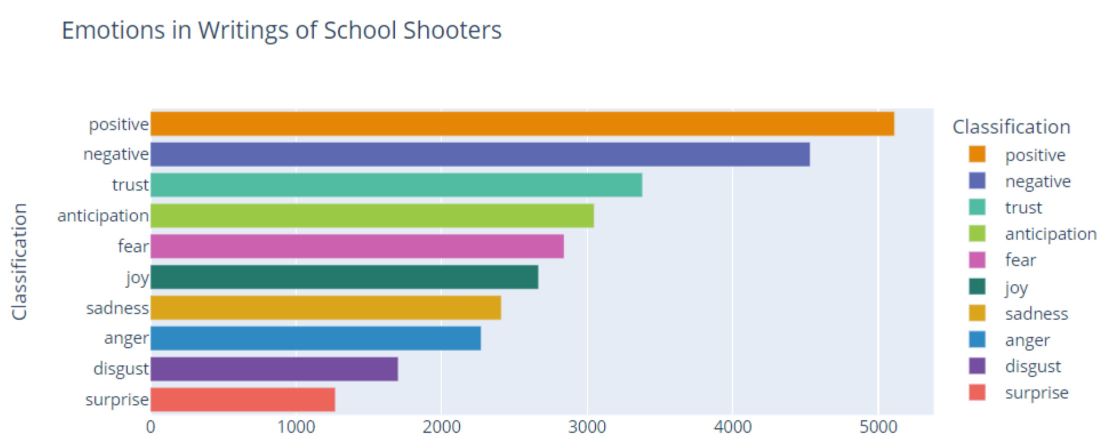


Figure 10

	Classification	Count
3	positive	5111
1	negative	4532
4	trust	3381
2	anticipation	3048
6	fear	2842
5	joy	2667
7	sadness	2411
8	anger	2273
0	disgust	1703
9	surprise	1271

Table 1

Here we see that positivity is the highest emotion detected within the writings of the school shooters, followed by negativity, trust, anticipation, fear, joy, sadness, anger, disgust, and surprise. It's also worth mentioning that words can be counted for multiple emotions: for instance, we can see that the word "kill" is counted for fear, negativity, and sadness.

Figure 11

```
word = "kill"
```

```
text_input = NRCLex(word)
```

```
emoScore = text_input.raw_emotion_scores
```

```
print(emoScore)
```

```
{'fear': 1, 'negative': 1, 'sadness': 1}
```

We can also observe the emotional polarity of the dataset. This is scored on a scale from -1 to 1, where -1 indicates negative sentiment, 0 is neutral sentiment, and 1 is positive sentiment. Here, we can see the distribution of sentiment polarity of both the bloggers and shooters. The distribution for the shooters favors positive values rather than negatives ones. Most of the distribution is between -0.2 and 0.2, where anything less than or greater than those respective values are considered outliers.

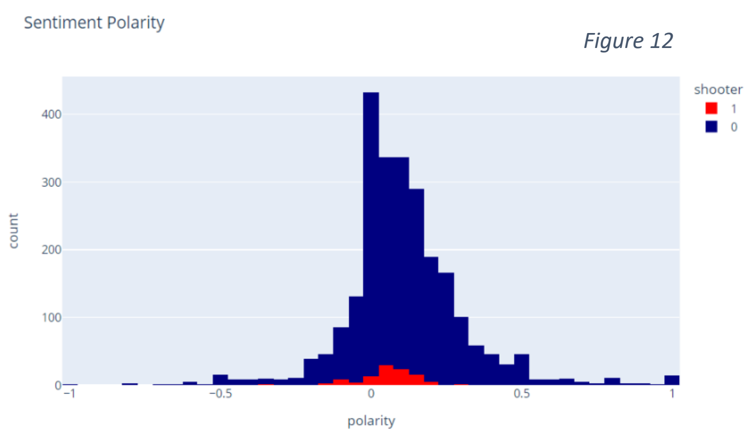


Figure 12

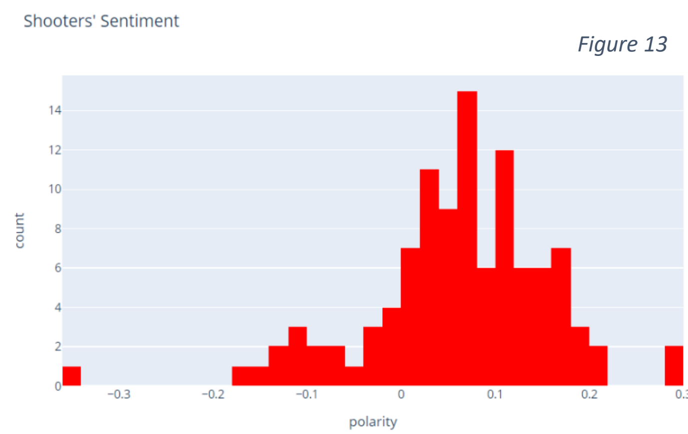


Figure 13

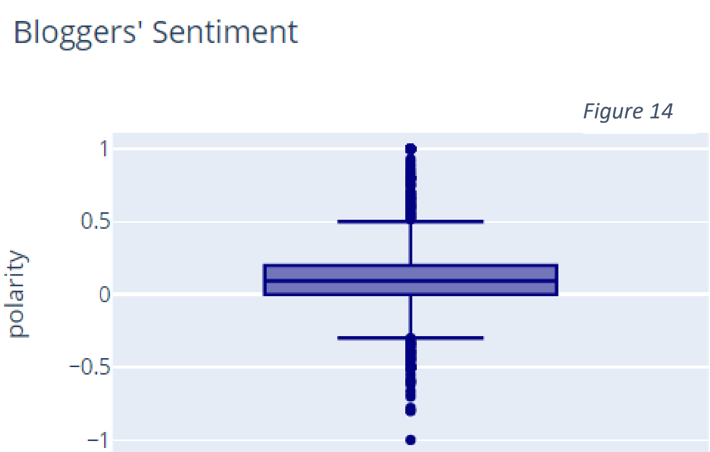


Figure 14

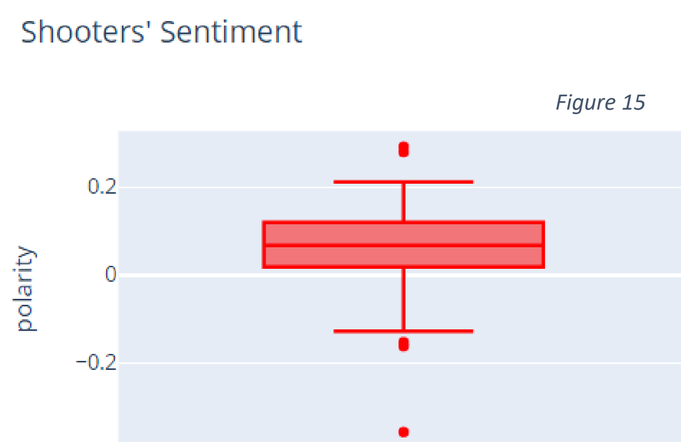


Figure 15

Splitting and Preparing for Training and Testing

The dataset is split into training and testing sets with 70% and 30% of the data respectively. Both sets are then tagged to clearly indicate whether each text or writing is by a shooter (1) or blogger (0). This essentially associates the text with the *shooter* column where the information is stored into its own version of a list. Next, a PV-DBOW model was created in doc2vec on the tagged training set. The neural net was trained on 30 iterations to create a vector size of 300. From this, a vocabulary was built for the training set and a final vector feature for the model. The final testing and training sets are then re-processed and updated so that they can be evaluated with the new transformations that took place. Finally, to address the issue of the imbalance in shooters and bloggers as witnessed in the percentage of writings by shooters (<5%) and in figures 1, 2, and 5, the shooters are then oversampled via an adaptive synthetic algorithm (ADASYN) to increase the number of shooters in the training set. By "oversampled", this means the minor class makes copies of itself.

Classifier Performance and Evaluation

In all, six different classifiers were evaluated to identify school shooters according to their writings: Logistic Regression, Polynomial Support Vector Machine (SVM), Linear SVM, Gradient Boost, Decision Tree, and a Multilayer Perceptron Neural Network (MLPNNet). Upon testing, the F1 results for the shooters were relatively low, with an average F1 test score of 0.29 or 29%. To improve these results, a distributed memory feature was added to the modelling. This is used to help guess the output or target writings from neighboring writings in tandem with the tagged documents, or in other words, it provides context to the document as to what is being guessed. For the DM feature, another doc2vec neural network model was instantiated and trained for 30 iterations to make a vector size of 300 words. All classifiers were trained and tested again. The average F1 test score of the shooters was 0.44 or 44%. For a breakdown of the F1 scores of all models in both cases, the following table is provided:

Table 2: Predictive Modelling Results

Models	PV-DBOW F1 Score	PV-DBOW + DM F1 Score
Logistic Regression	0.29	0.75
SVM Gauss: Polynomial	0.28	0.27
SVM Gauss: Linear	0.26	0.57
Gradient Boost	0.34	0.4
Decision Tree	0.3	0.26
Multilayer Perceptron Nnet	0.27	0.7

Logistic Regression and the Multilayer Perceptron Neural Network had the most significant improvements in their performances. Gradient Boost had a worthy improvement in its performance as well. However, SVM Polynomial and the Decision Tree models performed worse as it seems the distributed memory diminished their testing results. The results are particularly desirable for the logistic regression as well as MLPNNet classifiers in identifying the shooters in this dataset as they both equated or exceed 70%.

However, when increasing the scale of the bloggers in the dataset by 10 times for a grand total of over 6 million words and having a class distribution of approximately 99.66% bloggers and 0.44% shooters, the predictive performances of the models did not hold up to scale. Omitting the SVM Polynomial and Decision Tree classifiers due to their degradation in the last evaluation, the following results ensued:

Figure 3: Predictive Modelling Results with 10X larger set

Models	PV-DBOW F1 Score	PV-DBOW + DM F1 Score
Logistic Regression	0.06	0.11
SVM Gauss: Linear	0.26	0.12
Gradient Boost	0.08	0.3
Multilayer Perceptron Nnet	0.07	0.11

The SVM Linear model performed best overall without the distributed memory feature. With it, its performance was worse. Gradient Boost and Decision Tree followed a degradation trend as well. Like last time, Logistic Regression and the MLPNNet model performed better with DM and had similar increases in their performance. However, their performance was not as successful.

Conclusion and Recommendations

Looking at the results, there is some success in identifying school shooters by their digital writings. The most notable success was with the original dataset, and it seems quite plausible the percentage of shooters to bloggers played a big role as the classifiers' performance drastically diminished with a higher ratio of bloggers to shooters as shown in table 3. With the original dataset, the results were quite promising especially with the combination of the PV-DBOW and features.

Determining and detecting school shooters in the real world is likely to remain attributed to a multi-faceted approach that machine learning cannot replace but potentially enhance. Nothing beats the intuition and observation of a human being, and it's important to remain vigilant about those of whom may seem to fit the features of a school shooter, to which I defer to Dr. Peter Langman's research. Simply classifying texts and predicting if it is school shooter may not be enough. Due to the limited amount of data available regarding the writings of school shooters, it might be reasonable to approach this study in considering all predetermined mass shootings or terroristic plots in addition to school shooters. If anything, this study indicated that there is a linguistic pattern that distinguishes school shooters from non-school shooters that are identifiable under a certain proportion within the data. However, because this process entails transforming words into numbers and documents into complicated matrices, there is no interpretation to give a sensible distinction of a school shooter in a linguistical sense according to this study.

Harms, W. (n.d.). Psychopaths are not neurally equipped to have concern for others. Retrieved October 21, 2022, from https://news.uchicago.edu/story/psychopaths-are-not-neurally-equipped-have-concern-others

Jaynes, E. T., & Bretthorst, G. L. (2021). Probability theory the logic of Science. Cambridge: Cambridge Univ. Press.

Kiehl, K. A., & Hoffman, M. B. (2011). The Criminal Psychopath: History, Neuroscience, Treatment, and Economics. Jurimetrics: The Criminal Psychopath: History, Neuroscience, Treatment, and Economics, 51, 335.

Langman, P. (2013, October 18). School Shooters - Inside Their Minds (E. Shimer Bowers, Interviewer) [Review of School Shooters - Inside Their Minds]. In Lehigh University. https://ed.lehigh.edu/news-events/news/school-shooters-inside-their-minds#:~:text=The%20latter%20things%20fall%20under%20a%20category%20called,warning%20signs%2C%20they%E2%80%99d%20know%20they%E2%80%99d%20better%20tell%20someone.

Langman, Dr. P. (2015, January 15). *School Shooters: There Is No Sound Bite Research highlights the diversity of perpetrators.* [Review of *School Shooters: There Is No Sound Bite Research highlights the diversity of perpetrators.*]. Psychology Today; Psychology Today. https://www.psychologytoday.com/us/blog/keeping-kids-safe/201501/school-shooters-there-is-no-sound-bite

Langman, P. (n.d.). School Shooters [Review of School Shooters]. School Shooters. Retrieved September 17, 2022, from https://schoolshooters.info/

Li, S. (2018, December 4). *Multi-Class Text Classification with Doc2Vec & Logistic Regression*. Medium. https://towardsdatascience.com/multi-class-text-classification-with-doc2vec-logistic-regression-9da9947b43f4

Male to Female Ratio by State 2022. (n.d.). Retrieved October 21, 2022, from https://worldpopulationreview.com/state-rankings/male-to-female-ratio-by-state

Mass Shootings: Preparing for Violent Events. (2019, July). Retrieved October 19, 2022, from https://cops.usdoj.gov/html/dispatch/07-2019/mass_shootings.html

Meloy, J. R., & O'Toole, M. E. (2011). The Concept of Leakage in Threat Assessment. Behavioral Sciences & the Law, 29(4), 513–527. https://doi.org/10.1002/bsl.986

Merriam-Webster. (n.d.). Leak. In Merriam-Webster.com dictionary. Retrieved September 20, 2022, from https://www.merriam-webster.com/dictionary/leak

Neuman, Y. (2012). On revenge. Psychoanalysis, Culture & Society, 17(1), 1-15. doi:10.1057/pcs.2012.4

Neuman, Y., & Cohen, Y. (2014). A Vectorial Semantics Approach to Personality Assessment. Scientific Reports, 4(1). https://doi.org/10.1038/srep04761

Neuman, Y., Assaf, D., Cohen, Y., & Knoll, J. L. (2015). Profiling School Shooters: Automatic Text-Based Analysis. Frontiers in Psychiatry, 6. https://doi.org/10.3389/fpsyt.2015.00086

NRCLex. (2022, August 31). PyPI. https://pypi.org/project/NRCLex/

O'Toole, M. E. (n.d.). The School Shooter: A Threat Assessment Perspective [Review of The School Shooter: A Threat Assessment Perspective]. Critical Incident Response Group, National Center for the Analysis of Violent Crime, FBI Academy, 11–25. Retrieved September 16, 2022, from https://www.fbi.gov/file-repository/stats-services-publications-school-shooter-school-shooter/view

Purgato, V. P. (2021, December 29). *How To Easily Extract Text From Any PDF With Python*. Medium. https://medium.com/analytics-vidhya/how-to-easily-extract-text-from-any-pdf-with-python-fc6efd1dedbe

Qingwan. (2021, September 11). *NLP-Emotion-Detection/emotion_detection_nrclex.ipynb at main · xiaoqingwan/NLP-Emotion-Detection*. GitHub. Retrieved November 5, 2022, from https://github.com/xiaoqingwan/NLP-Emotion-Detection/blob/main/emotion_detection_nrclex.ipynb

Quoc V. Le, & Tomas Mikolov. (2014). Distributed Representations of Sentences and Documents. *International Conference on Machine Learning*, *4*, 1188–1196.

Rich, S. (2022, May 21). School Shootings. Washington Post.

Smart, R., & Schell, T. L. (2021, April 15). Mass shootings in the United States. Retrieved October 21, 2022, from https://www.rand.org/research/gun-policy/analysis/essays/mass-shootings.html

United States Secret Service, & National Threat Assessment Center. (2019). Protecting America's Schools: A US Secret Service Analysis of Targeted School Violence [Review of Protecting America's Schools: A US Secret Service Analysis of Targeted School Violence]. https://www.secretservice.gov/sites/default/files/2020-04/Protecting_Americas_Schools.pdf

Woodrow Cox, J., Rich, S., Chiu, A., Thacker, H., Chong, L., Muyskens, J., & Ulmanu, M. (2022, May 27). More than 311,000 students have experienced gun violence at school since Columbine [Review of More than 311,000 students have experienced gun violence at school since Columbine]. Washington Post; Washington Post. https://www.washingtonpost.com/graphics/2018/local/school-shootings-database/

** The Washington Post considers students exposed to gun violence as those who were in the same facility of where the incident has occurred.*

Appendix: A

Classifiers: 96% Bloggers, 4% Shooters

```
import pandas as pd
import numpy as np
from tqdm import tqdm
tqdm.pandas(desc="progress-bar")
from gensim.models import Doc2Vec
from sklearn import utils
from sklearn.model_selection import train_test_split
import gensim
from sklearn.linear_model import LogisticRegression
from gensim.models.doc2vec import TaggedDocument
import re
import seaborn as sns
import matplotlib.pyplot as plt
from textblob import TextBlob
```

```
df = pd.read_csv("bloggers_and_shooters_final_final_final.csv", encoding = 'l
atin1')
del df['Unnamed: 0']
```

```
df = df.dropna()
```

```
df.head()
```

author

topic

text

shooter

0

WellingtonDeOliveira

Suicide Note

You should first know that the impure cannot t...

1

1

SeunHuiCho

Manifesto

Oh the happiness I could have had mingling amo...

1

2

SeunHuiCho

Manifesto

we will hunt you down, you Lovers of Terrorism...

1

3

SeunHuiCho

Manifesto

Now that the slate has been cleaned and you ha...

1

4

RobertButler

Note

Everybody that used to know me I�m sry but Oma...

1

```
df.shape
```

```
(2470, 4)
```

```
# Word count:


df.index = range(2470)

count = df.text.apply(lambda x: len(x.split(' '))).sum()
print("There are ", count, " words in this dataset")
```

```
There are  674492  words in this dataset
```

```
print(df.describe)
print(df.dtypes)
```

```
<bound method NDFrame.describe of                      author          topic  \
0     WellingtonDeOliveira  Suicide Note
1               SeunHuiCho     Manifesto
2               SeunHuiCho     Manifesto
3               SeunHuiCho     Manifesto
```

```
4              RobertButler          Note
...                     ...           ...
2465                1000866       Student
2466                1000866       Student
2467                1000866       Student
2468                1000866       Student
2469                1000866       Student

                                                    text  shooter
0     You should first know that the impure cannot t...        1
1     Oh the happiness I could have had mingling amo...        1
2     we will hunt you down, you Lovers of Terrorism...        1
3     Now that the slate has been cleaned and you ha...        1
4     Everybody that used to know me I'm sry but Oma...        1
...                                                 ...      ...
2465         The only people I'd even want to talk t...        0
2466          OK, so, I wanted to change the little ...        0
2467         So, yesterday was alright. This whole w...        0
2468         So, today was a good day. I read for th...        0
2469         Yeah, so...wow. I'm a bit stressed out ...        0

[2470 rows x 4 columns]>
author      object
topic       object
text        object
shooter      int64
dtype: object
```

```
# df.loc[(df.Product == p_id)

# shooter_only = df2.loc[(df2.shooter == "1")]
# blogger_only = df2[(df2['shooter'] == "0")]

shooters_only = df.loc[(df.shooter == 1)]
bloggers_only = df.loc[(df.shooter == 0)]

print("Percentage of words that are written by shooters: ", len(shooters_only
.text) / len(df.text) * 100, "%")
print("Percentage of words written by bloggers: ", len(bloggers_only.text)/le
n(df.text) * 100, "%")
```

```
Percentage of words that are written by shooters:  4.291497975708502 %
Percentage of words written by bloggers:  95.70850202429149 %
```

As we can see above, just over 4% of writings in this data is by school shooters. Oversampling will introduce more observations of this class, the minority class. This is to be performed after the data is split into testing/training

PreProcessing Text

```
import string
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer

from nltk import pos_tag

df.shooter = df.shooter.astype(str)

def LiteCleaning(t):

    # Lower case text
    t = t.lower()

    # Removing Whitespace
    def remove_whitespace(t):
        return " ".join(t.split())
    t = remove_whitespace(t)

    # Removing punctuations
    punctuations = string.punctuation
    punctuations = punctuations + string.digits + "’" + '“”' + '–' + "‘" + "'
"
    table_ = str.maketrans(" ", " ", punctuations)
    t = t.translate(table_)

    tokenize = word_tokenize(t)

    def to_list(string):
        li = []
        li[:0] = string
        return li
    tokenize = to_list(tokenize)

    def lemmatizer(tokenize):
        wordnet = WordNetLemmatizer()
        lemWords = [wordnet.lemmatize(tokenized) for tokenized in tokenize]
        return lemWords
    lemmed = lemmatizer(tokenize)

       # Removing stop words
    stop = 'em', "'", "wa","n'", "ha", "didn", 'male','www.schoolshooters.inf
o', 'peter', 'langman', 'phd', 'version', 'january', 'february', 'march', 'ap
ril', 'may', 'june', 'july', 'august', 'september', 'october','november', 'de
cember', '2013', '2012', '2011', '2014', '2017', '2016', '2018', '2019', '202
0', '2010', '2009','2008', '2007', '2006', '2005', '2004', '2003', '2002', '2
001', '2000', '1999', 'bla', 'u', 'yo', 'youre', 'aint', 'ive', 'female', 'im
', 'didnt', 'like', 'dont', 'see', 'isnt', 'whenever', 'dont', 'cant', 'way',
'want', 'around', 'everything', 'could', 'become', 'show', 'others', 'see', '
```

```
something', 'else', 'make', 'fall', 'often', 'get', 'go', 'take', 'may', 'muc
h', 'anyone', 'ever', 'let', 'try', 'tell', 'give', 'get', 'me-by', 'me-if',
'act','i',  'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves
', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', '
it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves'
, 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those'
, 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', '
if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with'
, 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'afte
r', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when',
'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', '
other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'tha
n', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should',
"should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "are
n't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "ha
dn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'should
n', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'would
n', "wouldn't"

    final = [l for l in lemmed if not l in stop]

    tag_noun = [l for l, pos in pos_tag(final) if pos.startswith("N")]
    tag_verb = [l for l, pos in pos_tag(final) if pos.startswith("V")]
    tag_adj = [l for l, pos in pos_tag(final) if pos.startswith("J")]

    words = tag_noun + tag_verb + tag_adj


    return words

df["text"] = df["text"].apply(LiteCleaning)

df.head()
```

author

topic

text

shooter

0

WellingtonDeOliveira

Suicide Note

[impure, touch, chaste, chastity, marriage, gl...

1

1

SeunHuiCho

Manifesto

[happiness, hedonist, nof, didn�t, shit, ask, ...

1

2

SeunHuiCho

Manifesto

[hunt, terrorism, wan, pretend, devout, wan, d...

1

3

SeunHuiCho

Manifesto

[slate, attention, question, nare, truth, miss...

1

4

RobertButler

Note

[everybody, sry, school, gon, shit, school, wo...

1

```
import os as os
```

```
os.getcwd()
```

```
'C:\\Users\\\\ '
```

Train-Testing split, Oversampling

```
# Splitting
train, test = train_test_split(df, test_size = 0.30, random_state = 57)
```

```
test.head()
```

author

topic

text

shooter

1504

3026701

indUnk

[dangeresque, year, id, talk, right, distro, h…

0

1506

3022585

Education

[sex, city, tb, people, girl, love]

0

1109

3503162

Student

[test, today, drama, friday, semester, ok, thi…

0

245

788927

Communications-Media

[band, relient, k, jeremy, camp, anberlin, sup…

0

110

988941

Student

[cheese]

0

```
# import nltk
# from nltk.corpus import stopwords

# def tokenize_text(t):
#     tokens = []
#     for sent in nltk.sent_tokenize(t):
#         for word in nltk.word_tokenize(sent):
#             if len(word) <2:
#                 continue
#                 tokens.append(word.lower())
#             return tokens


train_tagged = train.apply(lambda r:
                           TaggedDocument(words = (r["text"]),
                                          tags = [r.shooter]), axis = 1)

test_tagged = test.apply(lambda r:
                         TaggedDocument(words = (r["text"]),
                                        tags = [r.shooter]), axis = 1)
```

Distributed Bag of Words

```
import multiprocessing

cores = multiprocessing.cpu_count()

# Vocabulary

model_dbow = Doc2Vec(dm= 0, vector_size =300,
                     negative = 5, hs = 0,
                     min_count = 2, sample = 0,
                     workers = cores)

model_dbow.build_vocab([x for x in tqdm(train_tagged.values)])

100%|██████████| 1729/1729 [00:00<00:00, 2311010.71it/s]
```

Training Doc2Vec model

30 epochs

```
%%time
for epoch in range(30):
    model_dbow.train(utils.shuffle([x for x in tqdm(train_tagged.values)]),
                     total_examples = len(train_tagged.values),
                     epochs =1)
    model_dbow.alpha -= 0.002
    model_dbow.min_alpha = model_dbow.alpha
```

```
100%|██████████| 1729/1729 [00:00<00:00, 2316177.46it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2316177.46it/s]
100%|██████████| 1729/1729 [00:00<?, ?it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2315437.94it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2324343.47it/s]
100%|██████████| 1729/1729 [00:00<?, ?it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2315437.94it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2305134.02it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2319140.27it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2315437.94it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2312484.57it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2315437.94it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2315437.94it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2319140.27it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2315437.94it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2316177.46it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2319882.15it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2314698.89it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2316177.46it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2316917.45it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2318398.85it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2316177.46it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2316177.46it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2317657.91it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2316177.46it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2317657.91it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2313960.31it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2274051.93it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2317657.91it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2317657.91it/s]


CPU times: total: 14 s
Wall time: 3.62 s
```

```python
from sklearn.metrics import confusion_matrix,roc_curve,roc_auc_score,accuracy
_score, plot_confusion_matrix,classification_report

from collections import Counter

def vec_for_learning(model, tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0],
                                 model.infer_vector(doc.words))
                                for doc in sents])

    return targets, regressors


# vec_for_learning(model, tagged_docs):
```

```
#     sents = tagged_docs.values
#     targets, regressors = zip(*[(doc.tags[0], model.infer_vector(doc.words,
steps=20)) for doc in sents])
#     return targets, regressors

# def vec_for_learning(model, tagged_docs):
#     sents = tagged_docs.values
#     targets, regressors = zip(*[(doc.tags[0],
#                                   model.infer_vector(doc.words, steps = 30))
#                                  for doc in sents])
#     return targets, regressors

# Train-Test Split

y_train, X_train = vec_for_learning(model_dbow, train_tagged)
y_test, X_test = vec_for_learning(model_dbow, test_tagged)
```

Oversampling

Balancing the training classes

```
from imblearn.over_sampling import SMOTE, ADASYN
from imblearn.combine import SMOTEENN
from imblearn.pipeline import make_pipeline

X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)

print(sorted(Counter(y_resampled).items()))

[('0', 1651), ('1', 1653)]
```

Classifiers

```
from sklearn.metrics import make_scorer
```

Logistic Classifier

```
logReg = LogisticRegression(n_jobs = 5, C = 1e5, random_state = 0)
logReg.fit(X_resampled, y_resampled)
y_pred_logit = logReg.predict(X_test)

logi_report = classification_report(y_test, y_pred_logit)
logi_conf_mat = confusion_matrix(y_test, y_pred_logit)

print(f"Logistic Regression Testing Results:\n", logi_report)
print(f"Confusion Matrix:\n", logi_conf_mat)

Logistic Regression Testing Results:
               precision    recall  f1-score   support

           0       0.99      0.89      0.94       713
           1       0.22      0.75      0.34        28

    accuracy                           0.89       741
```

```
   macro avg       0.60      0.82      0.64       741
weighted avg       0.96      0.89      0.92       741

Confusion Matrix:
 [[637  76]
 [  7  21]]
```

Support Vector Machine

```
from sklearn.svm import SVC
```

SVM Polynomial

```
svc_gauss = SVC(kernel = 'poly', random_state = 0)
svc_gauss.fit(X_resampled, y_resampled)
y_pred = svc_gauss.predict(X_test)

svm_report = classification_report(y_test, y_pred)
svm_conf_mat = confusion_matrix(y_test, y_pred)

print(f"SVM Testing Results: \n", svm_report)
print(f"Confusion Matrix \n", svm_conf_mat)
```

```
SVM Testing Results:
               precision    recall  f1-score   support

           0       0.99      0.82      0.90       713
           1       0.16      0.86      0.27        28

    accuracy                           0.82       741
   macro avg       0.58      0.84      0.58       741
weighted avg       0.96      0.82      0.88       741

Confusion Matrix
 [[586 127]
 [  4  24]]
```

SVM Linear

```
svc_gauss = SVC(kernel = 'linear', random_state = 0)
svc_gauss.fit(X_resampled, y_resampled)
y_pred = svc_gauss.predict(X_test)

svm_report = classification_report(y_test, y_pred)
svm_conf_mat = confusion_matrix(y_test, y_pred)

print(f"SVM Testing Results: \n", svm_report)
print(f"Confusion Matrix \n", svm_conf_mat)
```

```
SVM Testing Results:
               precision    recall  f1-score   support

           0       0.99      0.78      0.87       713
           1       0.13      0.86      0.23        28
```

```
    accuracy                           0.78       741
   macro avg       0.56      0.82      0.55       741
weighted avg       0.96      0.78      0.85       741

Confusion Matrix
 [[557 156]
 [  4  24]]
```

Gradient Boosting Classifier

```
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)

from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier(n_estimators = 100, learning_rate = 0.5,
                                max_depth =2, random_state = 0)
gbc.fit(X_resampled, y_resampled)

y_pred = gbc.predict(X_test)

gbc_report = classification_report(y_test, y_pred)
gbc_conf_mat = confusion_matrix(y_test, y_pred)

print(f"Gradient Boosting Report: \n",gbc_report)
print(f"Confusion Matrix: \n", gbc_conf_mat)
```

```
Gradient Boosting Report:
               precision    recall  f1-score   support

           0       0.98      0.93      0.95       713
           1       0.22      0.50      0.30        28

    accuracy                           0.91       741
   macro avg       0.60      0.71      0.63       741
weighted avg       0.95      0.91      0.93       741

Confusion Matrix:
 [[663  50]
 [ 14  14]]
```

Decision Tree

```
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)

# Decision Tree

from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(random_state = 0)
classifier.fit(X_resampled, y_resampled)
y_pred = classifier.predict(X_test)
```

```
DTC_report = classification_report(y_test, y_pred)
DTC_conf_mat = confusion_matrix(y_test, y_pred)

print(f"Decision Tree Classification Report: \n", DTC_report)
print(f"Decision Tree Confusion Matrix: \n", DTC_conf_mat)
```

```
Decision Tree Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.94      0.96       713
           1       0.23      0.50      0.32        28

    accuracy                           0.92       741
   macro avg       0.61      0.72      0.64       741
weighted avg       0.95      0.92      0.93       741

Decision Tree Confusion Matrix:
 [[667  46]
 [ 14  14]]
```

Nueral Network: Multilayer Perceptron Classifier

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=6000, random_st
ate = 0)
mlp.fit(X_resampled, y_resampled)

y_pred = mlp.predict(X_test)

MLP_report = classification_report(y_test, y_pred)
MLP_conf_mat = confusion_matrix(y_test, y_pred)

print(f"Multi-Layer Perceptron Classification Report: \n", MLP_report)
print(f"Multi-Layer Perceptron Classification Confusion Matrix: \n", MLP_conf
_mat)
```

```
Multi-Layer Perceptron Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.81      0.89       713
           1       0.15      0.86      0.25        28

    accuracy                           0.81       741
   macro avg       0.57      0.83      0.57       741
weighted avg       0.96      0.81      0.87       741

Multi-Layer Perceptron Classification Confusion Matrix:
 [[574 139]
 [  4  24]]
```

Distributed Memory

this allows the model to use a memory to remember what is missing from the current context of the paragraph. This will represent the concept of the document. By instantiating a new Doc2Vec model, we will be able to combine the distributed memory with the distributed bag of words model, hopefully improving the prediction scores

```
model_dmm = Doc2Vec(dm=1, dm_mean = 1, vector_size = 300, window = 10, negati
ve  =5,
                    min_count = 1, workers = 5, alpha = 0.065, min_alpha = 0.
065)

model_dmm.build_vocab([x for x in tqdm(train_tagged.values)])
```

```
100%|██████████| 1729/1729 [00:00<00:00, 2314698.89it/s]
```

```
%%time

for epoch in range(30):
    model_dmm.train(utils.shuffle([x for x in tqdm(train_tagged.values)]),
                    total_examples = len(train_tagged.values), epochs = 1)
    model_dmm.alpha -= 0.002
    model_dmm.min_alpha = model_dmm.alpha
```

```
100%|██████████| 1729/1729 [00:00<00:00, 2316177.46it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2267652.16it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2313222.21it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2315437.94it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2312484.57it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2311010.71it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2308803.44it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2317657.91it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2322110.67it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2315437.94it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2311010.71it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2314698.89it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2320624.52it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2269071.22it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2312484.57it/s]
100%|██████████| 1729/1729 [00:00<?, ?it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2312484.57it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2284079.25it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2312484.57it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2314698.89it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2315437.94it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2313222.21it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2319140.27it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2396547.13it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2317657.91it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2318398.85it/s]
```

```
100%|██████████| 1729/1729 [00:00<?, ?it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2332567.26it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2314698.89it/s]
100%|██████████| 1729/1729 [00:00<00:00, 2313960.31it/s]


CPU times: total: 21.6 s
Wall time: 6.64 s
```

Model Pairing

```
from gensim.test.test_doc2vec import ConcatenatedDoc2Vec

new_model = ConcatenatedDoc2Vec([model_dbow, model_dmm])

def get_vectors(model, tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0],
                                 model.infer_vector(doc.words)) for doc in se
nts])
    return targets, regressors
```

Final Model Evaluations

```
y_train, X_train = get_vectors(new_model, train_tagged)
y_test, X_test = get_vectors(new_model, test_tagged)

# Incoporporating upsampling
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)

print(sorted(Counter(y_resampled).items()))

[('0', 1651), ('1', 1642)]
```

Logistic Regression

```
# Logistic Regression
# Training and Testing
logReg = LogisticRegression(n_jobs = 5, C = 1e5, random_state = 0)
logReg.fit(X_resampled, y_resampled)
y_pred = logReg.predict(X_test)

logi_report = classification_report(y_test, y_pred)
logi_conf_mat = confusion_matrix(y_test, y_pred)

print(f"Logistic Regression Testing Results:\n", logi_report)
print(f"Confusion Matrix:\n", logi_conf_mat)

Logistic Regression Testing Results:
               precision    recall  f1-score   support

           0       0.99      0.98      0.99       713
           1       0.66      0.82      0.73        28
```

```
    accuracy                           0.98       741
   macro avg       0.83      0.90      0.86       741
weighted avg       0.98      0.98      0.98       741

Confusion Matrix:
 [[701  12]
 [  5  23]]
```

SVM

Radial

```
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)

svc_gauss = SVC(kernel = 'rbf', random_state = 0)
svc_gauss.fit(X_resampled, y_resampled)
y_pred = svc_gauss.predict(X_test)

svm_report = classification_report(y_test, y_pred)
svm_conf_mat = confusion_matrix(y_test, y_pred)

print(f"SVM Testing Results: \n", svm_report)
print(f"Confusion Matrix \n", svm_conf_mat)
```

```
SVM Testing Results:
               precision    recall  f1-score   support

           0       0.99      0.83      0.90       713
           1       0.16      0.86      0.27        28

    accuracy                           0.83       741
   macro avg       0.58      0.84      0.59       741
weighted avg       0.96      0.83      0.88       741

Confusion Matrix
 [[589 124]
 [  4  24]]
```

Polynomial

```
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)

svc_gauss = SVC(kernel = 'poly', random_state = 0)
svc_gauss.fit(X_resampled, y_resampled)
y_pred = svc_gauss.predict(X_test)

svm_report = classification_report(y_test, y_pred)
svm_conf_mat = confusion_matrix(y_test, y_pred)

print(f"SVM Testing Results: \n", svm_report)
print(f"Confusion Matrix \n", svm_conf_mat)
```

```
SVM Testing Results:
               precision    recall  f1-score   support

           0       0.99      0.82      0.90       713
           1       0.16      0.86      0.27        28

    accuracy                           0.82       741
   macro avg       0.58      0.84      0.58       741
weighted avg       0.96      0.82      0.88       741

Confusion Matrix
 [[586 127]
 [  4  24]]
```

Linear

```
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)

svc_gauss = SVC(kernel = 'linear', random_state = 0)
svc_gauss.fit(X_resampled, y_resampled)
y_pred = svc_gauss.predict(X_test)

svm_report = classification_report(y_test, y_pred)
svm_conf_mat = confusion_matrix(y_test, y_pred)

print(f"SVM Testing Results: \n", svm_report)
print(f"Confusion Matrix \n", svm_conf_mat)

SVM Testing Results:
               precision    recall  f1-score   support

           0       0.99      0.94      0.97       713
           1       0.37      0.82      0.51        28

    accuracy                           0.94       741
   macro avg       0.68      0.88      0.74       741
weighted avg       0.97      0.94      0.95       741

Confusion Matrix
 [[673  40]
 [  5  23]]
```

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)

gbc = GradientBoostingClassifier(n_estimators = 100, learning_rate = 0.5,
                                max_depth =2, random_state = 0)
gbc.fit(X_resampled, y_resampled)
```

```
y_pred = gbc.predict(X_test)

gbc_report = classification_report(y_test, y_pred)
gbc_conf_mat = confusion_matrix(y_test, y_pred)

print(f"Gradient Boosting Report: \n",gbc_report)
print(f"Confusion Matrix: \n", gbc_conf_mat)
```

```
Gradient Boosting Report:
               precision    recall  f1-score   support

           0       0.96      1.00      0.98       713
           1       1.00      0.04      0.07        28

    accuracy                           0.96       741
   macro avg       0.98      0.52      0.53       741
weighted avg       0.96      0.96      0.95       741

Confusion Matrix:
 [[713   0]
 [ 27   1]]
```

Decision Tree

```
# Decision Tree

from sklearn.tree import DecisionTreeClassifier
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)

classifier = DecisionTreeClassifier()
classifier.fit(X_resampled, y_resampled)
y_pred = classifier.predict(X_test)

DTC_report = classification_report(y_test, y_pred)
DTC_conf_mat = confusion_matrix(y_test, y_pred)

print(f"Decision Tree Classification Report: \n", DTC_report)
print(f"Decision Tree Confusion Matrix: \n", DTC_conf_mat)
```

```
Decision Tree Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.96      0.96       713
           1       0.04      0.04      0.04        28

    accuracy                           0.93       741
   macro avg       0.50      0.50      0.50       741
weighted avg       0.93      0.93      0.93       741

Decision Tree Confusion Matrix:
```

```
[[687  26]
 [ 27   1]]
```

Neural Networks

```
# Nueral Network: Multilayer Perceptron Classifier
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)


from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=6000, random_st
ate = 0)
mlp.fit(X_resampled, y_resampled)

y_pred = mlp.predict(X_test)

MLP_report = classification_report(y_test, y_pred)
MLP_conf_mat = confusion_matrix(y_test, y_pred)

print(f"Multi-Layer Perceptron Classification Report: \n", MLP_report)
print(f"Multi-Layer Perceptron Classification Confusion Matrix: \n", MLP_conf
_mat)

Multi-Layer Perceptron Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.99      0.99       713
           1       0.81      0.79      0.80        28

    accuracy                           0.99       741
   macro avg       0.90      0.89      0.90       741
weighted avg       0.98      0.99      0.99       741

Multi-Layer Perceptron Classification Confusion Matrix:
 [[708   5]
 [  6  22]]
```

Because the nerual-net is the best performing model, it will be used for classifying authorship of the writers.

Appendix B

Classifiers with 99.6% Bloggers, 0.4% Shooters

```
import pandas as pd
import numpy as np
from tqdm import tqdm
tqdm.pandas(desc="progress-bar")
from gensim.models import Doc2Vec
from sklearn import utils
from sklearn.model_selection import train_test_split
import gensim
from sklearn.linear_model import LogisticRegression
from gensim.models.doc2vec import TaggedDocument
import re
import seaborn as sns
import matplotlib.pyplot as plt
from textblob import TextBlob

df = pd.read_csv("bloggers_and_shooters_final_final_final2.csv", encoding = 'latin1')
del df['Unnamed: 0']
df = df.dropna()

shooters_only = df.loc[(df.shooter == 1)]
bloggers_only = df.loc[(df.shooter == 0)]
print("Percentage of words that are written by shooters: ", len(shooters_only.text) / len(df.text) *
100, "%")
print("Percentage of words written by bloggers: ", len(bloggers_only.text)/len(df.text) * 100, "%")

import string
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer

from nltk import pos_tag
```

```
df.shooter = df.shooter.astype(str)

def LiteCleaning(t):

    # Lower case text
    t = t.lower()

    # Removing Whitespace
    def remove_whitespace(t):
        return " ".join(t.split())
    t = remove_whitespace(t)

    # Removing punctuations
    punctuations = string.punctuation
    punctuations = punctuations + string.digits + "’" + '“”' + '—' + "‘"
    table_ = str.maketrans(" ", " ", punctuations)
    t = t.translate(table_)

    tokenize = word_tokenize(t)

    def to_list(string):
        li = []
        li[:0] = string
        return li
    tokenize = to_list(tokenize)

    def lemmatizer(tokenize):
        wordnet = WordNetLemmatizer()
        lemWords = [wordnet.lemmatize(tokenized) for tokenized in tokenize]
        return lemWords
    lemmed = lemmatizer(tokenize)

      # Removing stop words
    stop = 'em', 'male','www.schoolshooters.info', 'peter', 'langman', 'phd', 'version', 'january',
```

```
'february', 'march', 'april', 'may', 'june', 'july', 'august', 'september', 'october','november',
'december', '2013', '2012', '2011', '2014', '2017', '2016', '2018', '2019', '2020', '2010', '2009','2008',
'2007', '2006', '2005', '2004', '2003', '2002', '2001', '2000', '1999', 'bla', 'u', 'yo', 'youre', 'aint', 'ive',
'female', 'im', 'didnt', 'like', 'dont', 'see', 'isnt', 'whenever', 'dont', 'cant', 'way', 'want', 'around',
'everything', 'could', 'become', 'show', 'others', 'see', 'something', 'else', 'make', 'fall', 'often', 'get',
'go', 'take', 'may', 'much', 'anyone', 'ever', 'let', 'try', 'tell', 'give', 'get', 'me-by', 'me-if', 'act','i',  'me',
'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours',
'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its',
'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that',
"that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at',
'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above',
'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once',
'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other',
'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just',
'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn',
"couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't",
'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't",
'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"

    final = [l for l in lemmed if not l in stop]

    tag_noun = [l for l, pos in pos_tag(final) if pos.startswith("N")]
    tag_verb = [l for l, pos in pos_tag(final) if pos.startswith("V")]
    tag_adj = [l for l, pos in pos_tag(final) if pos.startswith("J")]

    words = tag_noun + tag_verb + tag_adj


    return words

df["text"] = df["text"].apply(LiteCleaning)

train, test = train_test_split(df, test_size = 0.30, random_state = 0)
```

```
train_tagged = train.apply(lambda r:
                           TaggedDocument(words = (r["text"]),
                                          tags = [r.shooter]), axis = 1)

test_tagged = test.apply(lambda r:
                         TaggedDocument(words = (r["text"]),
                                        tags = [r.shooter]), axis = 1)

import multiprocessing
cores = multiprocessing.cpu_count()
# Vocabulary

model_dbow = Doc2Vec(dm= 0, vector_size =300,
                     negative = 5, hs = 0,
                     min_count = 2, sample = 0,
                     workers = cores)

model_dbow.build_vocab([x for x in tqdm(train_tagged.values)])

## Doc2VecModel
# %%time
for epoch in range(30):
    model_dbow.train(utils.shuffle([x for x in tqdm(train_tagged.values)]),
                     total_examples = len(train_tagged.values),
                     epochs =1)
    model_dbow.alpha -= 0.002
    model_dbow.min_alpha = model_dbow.alpha

from sklearn.metrics import confusion_matrix,roc_curve,roc_auc_score,accuracy_score,
plot_confusion_matrix,classification_report
from collections import Counter

def vec_for_learning(model, tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0],
```

```
                            model.infer_vector(doc.words))
                            for doc in sents])

    return targets, regressors

y_train, X_train = vec_for_learning(model_dbow, train_tagged)
y_test, X_test = vec_for_learning(model_dbow, test_tagged)

from imblearn.over_sampling import SMOTE, ADASYN
from imblearn.combine import SMOTEENN
from imblearn.pipeline import make_pipeline

X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)

from sklearn.metrics import make_scorer

## LOGIT
logReg = LogisticRegression(n_jobs = 5, C = 1e5, random_state = 0)
logReg.fit(X_resampled, y_resampled)
y_pred_logit = logReg.predict(X_test)

logi_report = classification_report(y_test, y_pred_logit)
logi_conf_mat = confusion_matrix(y_test, y_pred_logit)

print(f"Logistic Regression Testing Results DBOW:\n", logi_report)
print(f"Confusion Matrix:\n", logi_conf_mat)

## POLY SVM
from sklearn.svm import SVC
svc_gauss = SVC(kernel = 'poly', random_state = 0)
svc_gauss.fit(X_resampled, y_resampled)
y_pred = svc_gauss.predict(X_test)

svm_report = classification_report(y_test, y_pred)
svm_conf_mat = confusion_matrix(y_test, y_pred)
```

```
print(f"SVM Testing Results DBOW: \n", svm_report)
print(f"Confusion Matrix \n", svm_conf_mat)


## GRAD BOOST
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier(n_estimators = 100, learning_rate = 0.5,
                           max_depth =2, random_state = 0)
gbc.fit(X_resampled, y_resampled)

y_pred = gbc.predict(X_test)

gbc_report = classification_report(y_test, y_pred)
gbc_conf_mat = confusion_matrix(y_test, y_pred)

print(f"Gradient Boosting Report DBOW: \n",gbc_report)
print(f"Confusion Matrix: \n", gbc_conf_mat)

# Decision Tree
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(random_state = 0)
classifier.fit(X_resampled, y_resampled)
y_pred = classifier.predict(X_test)
DTC_report = classification_report(y_test, y_pred)
DTC_conf_mat = confusion_matrix(y_test, y_pred)
print(f"Decision Tree Classification Report DBOW: \n", DTC_report)
print(f"Decision Tree Confusion Matrix: \n", DTC_conf_mat)

# MLPNNET
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(100, 100, 100), max_iter=60000, random_state = 0)
mlp.fit(X_resampled, y_resampled)
```

```python
y_pred = mlp.predict(X_test)

MLP_report = classification_report(y_test, y_pred)
MLP_conf_mat = confusion_matrix(y_test, y_pred)

print(f"Multi-Layer Perceptron Classification Report DBOW: \n", MLP_report)
print(f"Multi-Layer Perceptron Classification Confusion Matrix: \n", MLP_conf_mat)


## DIST MEM
model_dmm = Doc2Vec(dm=1, dm_mean = 1, vector_size = 300, window = 10, negative  =5,
              min_count = 1, workers = 5, alpha = 0.065, min_alpha = 0.065)

model_dmm.build_vocab([x for x in tqdm(train_tagged.values)])

# %%time

for epoch in range(30):
   model_dmm.train(utils.shuffle([x for x in tqdm(train_tagged.values)]),
              total_examples = len(train_tagged.values), epochs = 1)
   model_dmm.alpha -= 0.002
   model_dmm.min_alpha = model_dmm.alpha

from gensim.test.test_doc2vec import ConcatenatedDoc2Vec

new_model = ConcatenatedDoc2Vec([model_dbow, model_dmm])

def get_vectors(model, tagged_docs):
   sents = tagged_docs.values
   targets, regressors = zip(*[(doc.tags[0],
                      model.infer_vector(doc.words)) for doc in sents])
   return targets, regressors
```

```
y_train, X_train = get_vectors(new_model, train_tagged)
y_test, X_test = get_vectors(new_model, test_tagged)

# Incoporporating upsampling
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)


# Logistic Regression
# Training and Testing
logReg = LogisticRegression(n_jobs = 5, C = 1e5, random_state = 0)
logReg.fit(X_resampled, y_resampled)
y_pred = logReg.predict(X_test)

logi_report = classification_report(y_test, y_pred)
logi_conf_mat = confusion_matrix(y_test, y_pred)

print(f"Logistic Regression Testing Results DBOW + DM:\n", logi_report)
print(f"Confusion Matrix:\n", logi_conf_mat)


## SVM LINEAR
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)

svc_gauss = SVC(kernel = 'linear', random_state = 0)
svc_gauss.fit(X_resampled, y_resampled)
y_pred = svc_gauss.predict(X_test)

svm_report = classification_report(y_test, y_pred)
svm_conf_mat = confusion_matrix(y_test, y_pred)

print(f"SVM Testing Results DBOW + DM: \n", svm_report)
print(f"Confusion Matrix \n", svm_conf_mat)


## GRAD BOOST
```

```python
from sklearn.ensemble import GradientBoostingClassifier
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)

gbc = GradientBoostingClassifier(n_estimators = 100, learning_rate = 0.5,
                                 max_depth =2, random_state = 0)
gbc.fit(X_resampled, y_resampled)


y_pred = gbc.predict(X_test)

gbc_report = classification_report(y_test, y_pred)
gbc_conf_mat = confusion_matrix(y_test, y_pred)

print(f"Gradient Boosting Report DBOW + DM: \n",gbc_report)
print(f"Confusion Matrix: \n", gbc_conf_mat)

# Decision Tree
from sklearn.tree import DecisionTreeClassifier
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)

classifier = DecisionTreeClassifier()
classifier.fit(X_resampled, y_resampled)
y_pred = classifier.predict(X_test)

DTC_report = classification_report(y_test, y_pred)
DTC_conf_mat = confusion_matrix(y_test, y_pred)

print(f"Decision Tree Classification Report DBOW + DM: \n", DTC_report)
print(f" Decision Tree Confusion Matrix: \n", DTC_conf_mat)


# Nueral Network: Multilayer Perceptron Classifier
X_resampled, y_resampled = ADASYN().fit_resample(X_train, y_train)
```

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 100), max_iter=6000, random_state = 0)
mlp.fit(X_resampled, y_resampled)

y_pred = mlp.predict(X_test)

MLP_report = classification_report(y_test, y_pred)
MLP_conf_mat = confusion_matrix(y_test, y_pred)

print(f"Multi-Layer Perceptron Classification Report DBOW + DM: \n", MLP_report)
print(f"Multi-Layer Perceptron Classification Confusion Matrix: \n", MLP_conf_mat)
```

```
Percentage of words that are written by shooters:  0.4462594198627542 %
Percentage of words written by bloggers:  99.55374058013724 %


100%|██████████| 16627/16627 [00:00<00:00, 3182380.79it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3182090.37it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783424.17it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2782868.82it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3182090.37it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2784202.04it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3167349.11it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2782757.78it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783090.93it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3181074.33it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3180494.03it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3180639.09it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3180639.09it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3180784.16it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3174702.63it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783202.00it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2474231.63it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3181074.33it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3180784.16it/s]
```

```
100%|██████████| 16627/16627 [00:00<00:00, 2784313.20it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783646.37it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783757.49it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3180639.09it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2474231.63it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3179623.97it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3180348.99it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783202.00it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3178319.78it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3180639.09it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3181800.01it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783424.17it/s]


Logistic Regression Testing Results DBOW:
              precision    recall  f1-score   support

           0       1.00      0.88      0.93      7095
           1       0.02      0.68      0.05        31

    accuracy                           0.88      7126
   macro avg       0.51      0.78      0.49      7126
weighted avg       0.99      0.88      0.93      7126

Confusion Matrix:
 [[6222  873]
 [  10   21]]
SVM Testing Results DBOW:
              precision    recall  f1-score   support

           0       1.00      0.86      0.92      7095
           1       0.02      0.81      0.05        31

    accuracy                           0.86      7126
   macro avg       0.51      0.83      0.49      7126
```

```
weighted avg      0.99      0.86      0.92      7126

Confusion Matrix
 [[6101  994]
 [   6   25]]
Gradient Boosting Report DBOW:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      7095
           1       0.03      0.16      0.05        31

    accuracy                           0.97      7126
   macro avg       0.51      0.57      0.52      7126
weighted avg       0.99      0.97      0.98      7126

Confusion Matrix:
 [[6934  161]
 [  26    5]]
Decision Tree Classification Report DBOW:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      7095
           1       0.01      0.03      0.02        31

    accuracy                           0.98      7126
   macro avg       0.50      0.51      0.50      7126
weighted avg       0.99      0.98      0.99      7126

Decision Tree Confusion Matrix:
 [[7011   84]
 [  30    1]]
Multi-Layer Perceptron Classification Report DBOW:
              precision    recall  f1-score   support

           0       1.00      0.91      0.95      7095
```

```
           1       0.03      0.65      0.06        31

    accuracy                           0.91      7126
   macro avg       0.51      0.78      0.51      7126
weighted avg       0.99      0.91      0.95      7126
```

Multi-Layer Perceptron Classification Confusion Matrix:

```
 [[6474  621]
 [  11   20]]
```

```
100%|██████████| 16627/16627 [00:00<00:00, 2783535.27it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2774675.44it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2784535.54it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783979.74it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2784202.04it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2784090.89it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2784869.12it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783202.00it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3181509.70it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783535.27it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2784090.89it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783535.27it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783090.93it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783535.27it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3182380.79it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783979.74it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2780760.50it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3181654.85it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3173691.30it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3181509.70it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783313.08it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783535.27it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783202.00it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2783535.27it/s]
```

```
100%|██████████| 16627/16627 [00:00<00:00, 2783979.74it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2769166.64it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2784646.73it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2784090.89it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2784090.89it/s]
100%|██████████| 16627/16627 [00:00<00:00, 2785536.53it/s]
100%|██████████| 16627/16627 [00:00<00:00, 3182235.57it/s]
```

```
Logistic Regression Testing Results DBOW + DM:
              precision    recall  f1-score   support

           0       1.00      0.88      0.93      7095
           1       0.03      0.97      0.06        31

    accuracy                           0.88      7126
   macro avg       0.52      0.92      0.50      7126
weighted avg       1.00      0.88      0.93      7126

Confusion Matrix:
 [[6224  871]
 [   1   30]]
SVM Testing Results DBOW + DM:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98      7095
           1       0.10      0.94      0.18        31

    accuracy                           0.96      7126
   macro avg       0.55      0.95      0.58      7126
weighted avg       1.00      0.96      0.98      7126

Confusion Matrix
 [[6829  266]
 [   2   29]]
```

Gradient Boosting Report DBOW + DM:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	7095
1	1.00	0.10	0.18	31
accuracy			1.00	7126
macro avg	1.00	0.55	0.59	7126
weighted avg	1.00	1.00	0.99	7126

Confusion Matrix:
[[7095 0]
[28 3]]

Decision Tree Classification Report DBOW + DM:

	precision	recall	f1-score	support
0	1.00	0.94	0.97	7095
1	0.00	0.03	0.00	31
accuracy			0.93	7126
macro avg	0.50	0.48	0.48	7126
weighted avg	0.99	0.93	0.96	7126

Decision Tree Confusion Matrix:
[[6651 444]
[30 1]]

Multi-Layer Perceptron Classification Report DBOW + DM:

	precision	recall	f1-score	support
0	1.00	0.97	0.99	7095
1	0.12	0.87	0.21	31
accuracy			0.97	7126
macro avg	0.56	0.92	0.60	7126
weighted avg	1.00	0.97	0.98	7126

Multi-Layer Perceptron Classification Confusion Matrix:

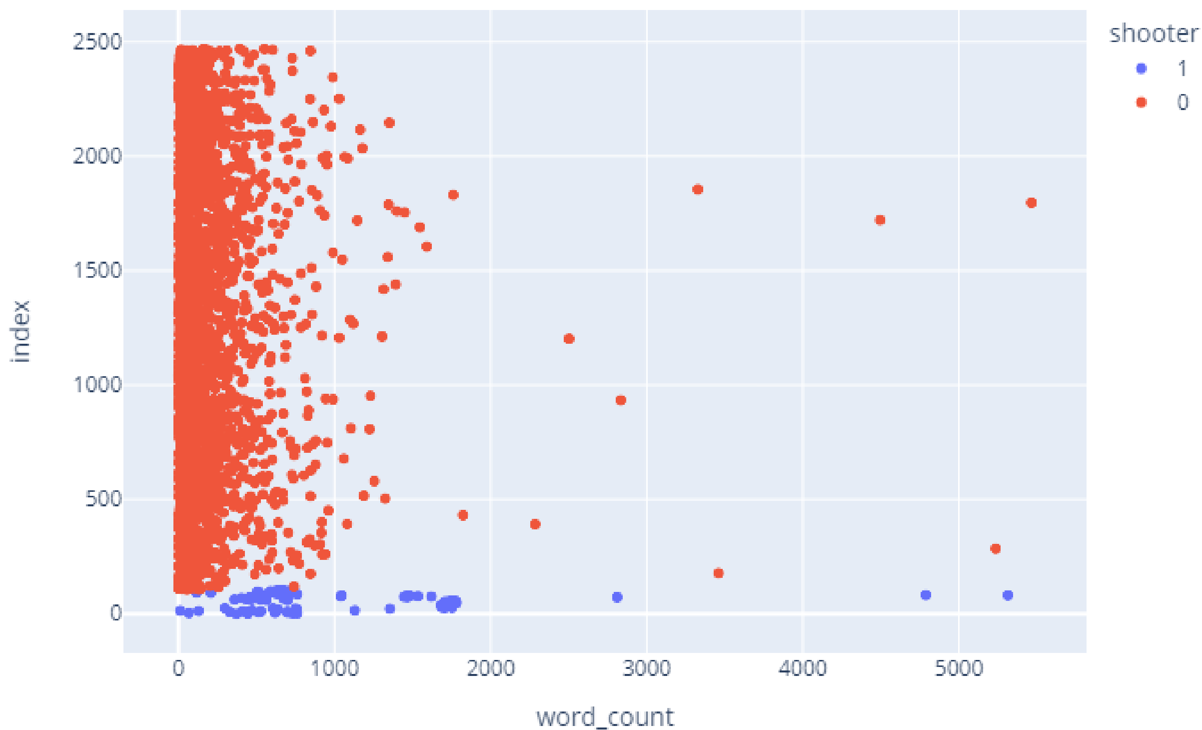
```
[[6899  196]
 [   4   27]]
```

Appendix: C

Visualizations

```
# Visualizations

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from textblob import TextBlob

from wordcloud import WordCloud

%matplotlib inline


df = pd.read_csv("bloggers_and_shooters_final_final_final.csv", encoding = 'latin1')

del df['Unnamed: 0']

df = df.dropna()


df['polarity'] = df['text'].map(lambda text: TextBlob(text).sentiment.polarity)

df['entry_len'] = df['text'].astype(str).apply(len)

df['word_count'] = df['text'].apply(lambda x: len(str(x).split()))


shooters_only_plot = df.loc[(df.shooter == 1)]

bloggers_only_plot = df.loc[(df.shooter == 0)]


df.shooter = df.shooter.astype(str)


import plotly.express as px


f5 = px.scatter(df, x = 'word_count', color = 'shooter',
```

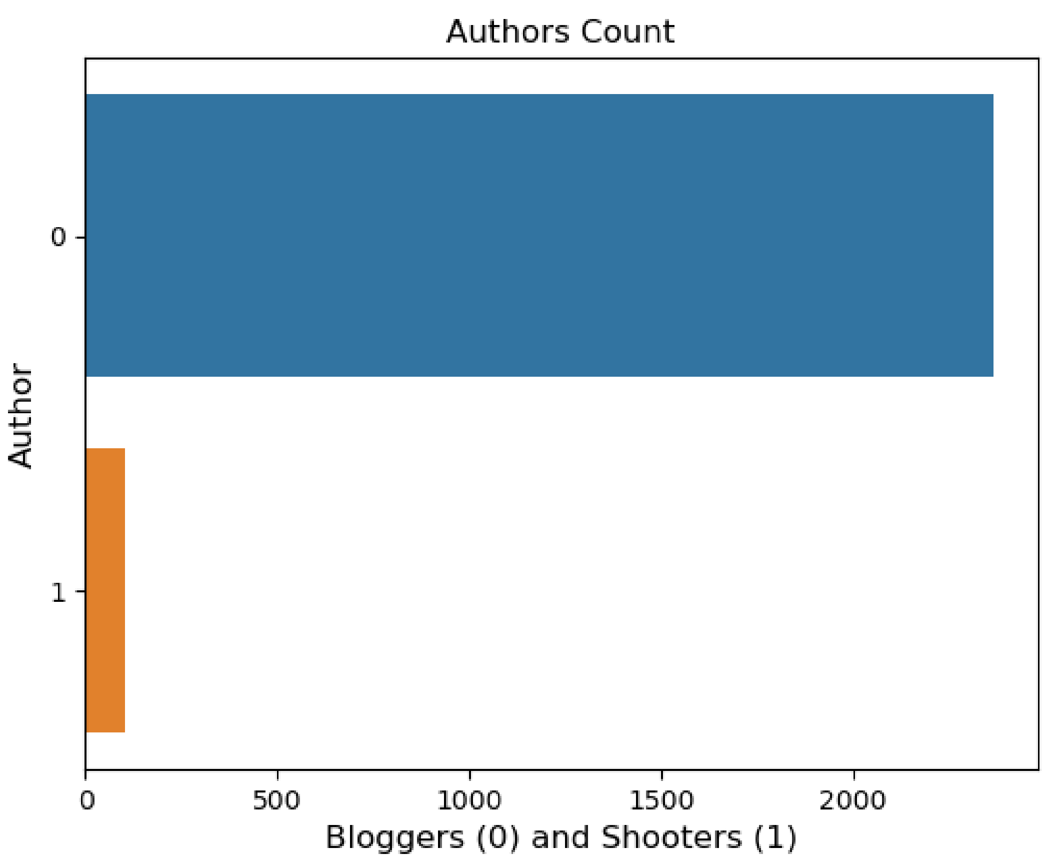
```
            color_continuous_scale = ['red', 'blue'],
             width = 800, height = 500)
f5.show()
```



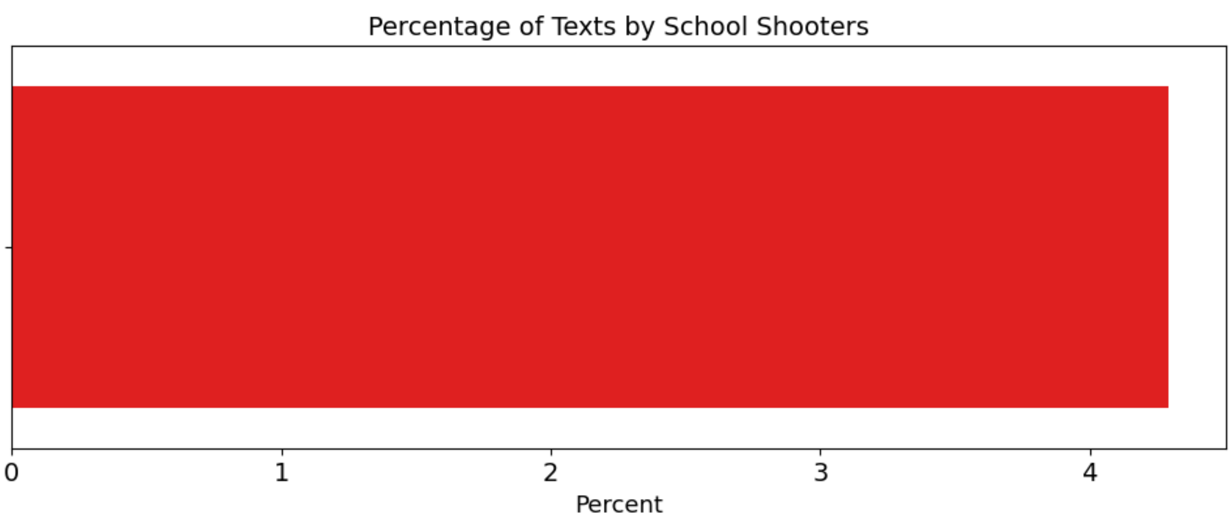

```
f6 = px.scatter(df, x = 'entry_len', color = 'shooter',
            color_discrete_sequence = ['green', 'purple'],
             width = 800, height = 500)
f6.show()
```


```
shooters_only = df.loc[(df.shooter == "1")]

bloggers_only = df.loc[(df.shooter == '0')]


x1 = len(shooters_only.text) / len(df.text)

x2 = len(bloggers_only.text)/len(df.text)


print("Percentage of words that are written by shooters: ", x1 * 100, "%")

print("Percentage of words written by bloggers: ", x2 * 100, "%")



df.index = range(2470)
```

```
count = df.text.apply(lambda x: len(x.split(' '))).sum()

print("There are ", count, " words in this dataset")


print(0.0429 * 247210, " words by school shooters")

print(0.957 * 247210, " words by bloggers")



sns.countplot(y = "shooter", data = df)

plt.xlabel("Bloggers (0) and Shooters (1)", fontsize = 12)

plt.ylabel("Author", fontsize = 12)

plt.title("Authors Count")

plt.show()
```



```
type_count = df["shooter"].value_counts()


plt.figure(figsize = (12,4))
sns.barplot(data = df*100, x= "shooter", errorbar = None, color = "red")
# plt.ylabel("Shooters", fontsize = 14)
plt.title("Percentage of Texts by School Shooters", fontsize = 14)
plt.xlabel("Percent", fontsize = 13)
plt.xticks(fontsize = 14)


plt.show();
```



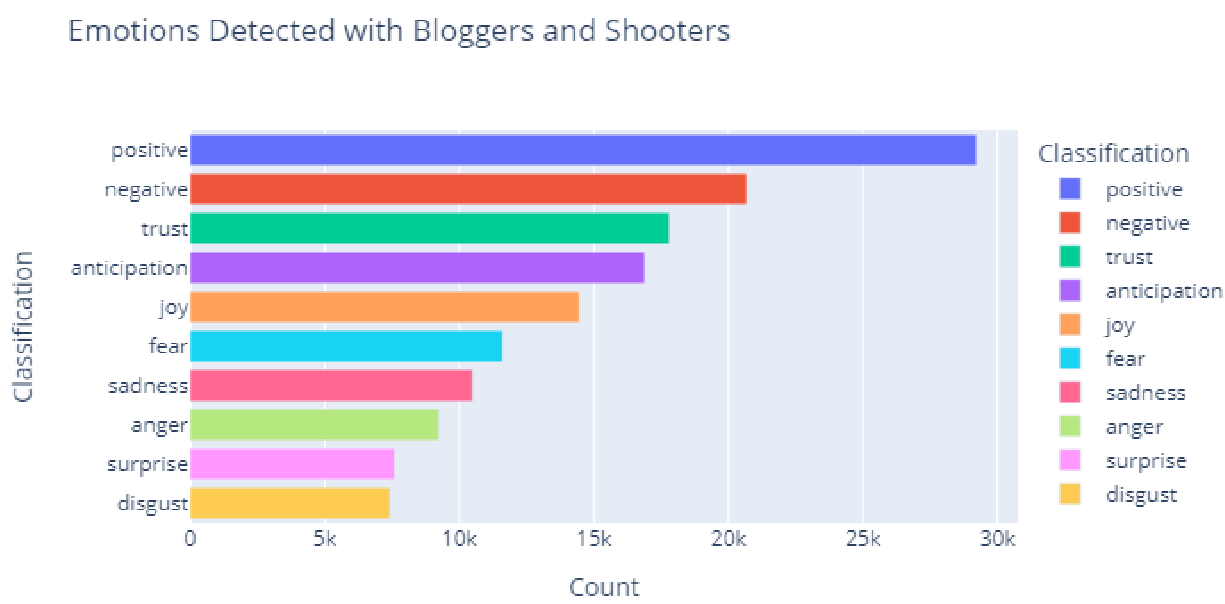

```
import string
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import pos_tag
from nrclex import NRCLex
```

```
def LiteCleaning(t):

    # Lower case text
    t = t.lower()

    # Removing Whitespace
    def remove_whitespace(t):
        return " ".join(''.join(t).split())
    t = remove_whitespace(t)

    # Removing punctuations
    punctuations = string.punctuation
    punctuations = punctuations + string.digits + "’" + '“”' + '—' + "‘"
    table_ = str.maketrans(" ", " ", punctuations)
    t = t.translate(table_)

    tokenize = word_tokenize(t)

    def to_list(string):
        li = []
        li[:0] = string
        return li
    tokenize = to_list(tokenize)

    def lemmatizer(tokenize):
        wordnet = WordNetLemmatizer()
        lemWords = [wordnet.lemmatize(tokenized) for tokenized in tokenize]
```

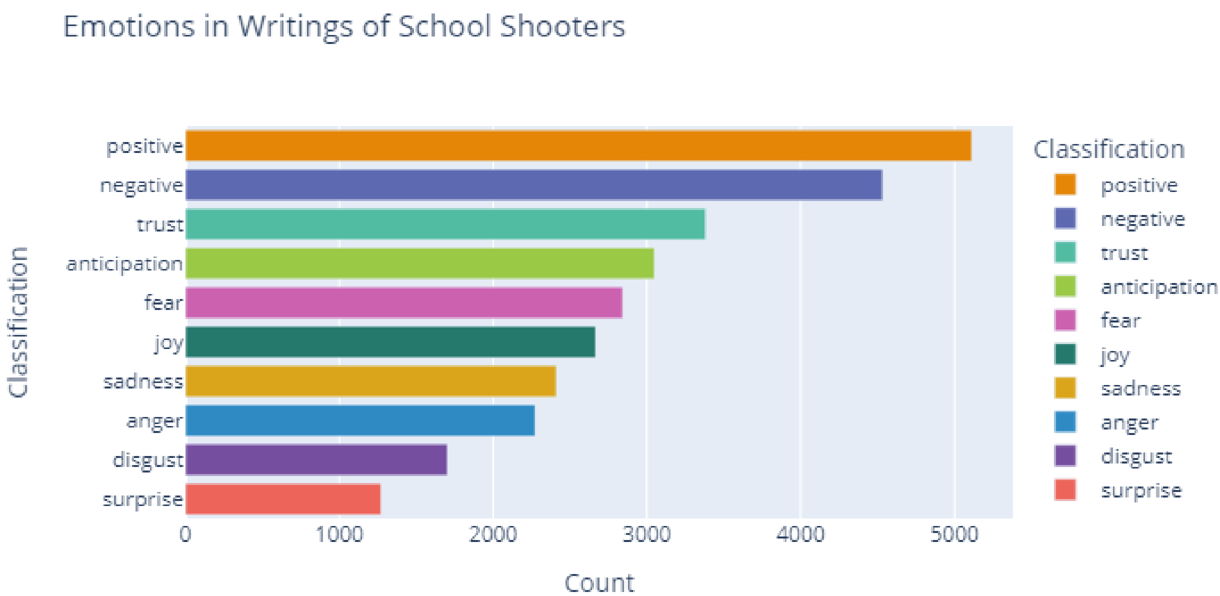
```
    return lemWords
  lemmed = lemmatizer(tokenize)


    # Removing stop words
  stop = 'em', 'male','www.schoolshooters.info', 'peter', 'langman', 'phd', 'version', 'january', 'february', 'march',
'april', 'may', 'june', 'july', 'august', 'september', 'october','november', 'december', '2013', '2012', '2011', '2014',
'2017', '2016', '2018', '2019', '2020', '2010', '2009','2008', '2007', '2006', '2005', '2004', '2003', '2002', '2001', '2000',
'1999', 'bla', 'u', 'yo', 'youre', 'aint', 'ive', 'female', 'im', 'didnt', 'like', 'dont', 'see', 'isnt', 'whenever', 'dont', 'cant',
'way', 'want', 'around', 'everything', 'could', 'become', 'show', 'others', 'see', 'something', 'else', 'make', 'fall', 'often',
'get', 'go', 'take', 'may', 'much', 'anyone', 'ever', 'let', 'try', 'tell', 'give', 'get', 'me-by', 'me-if', 'act','i',  'me', 'my',
'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself',
'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',
'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was',
'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then',
'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some',
'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should',
"should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn',
"doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',
"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won',
"won't", 'wouldn', "wouldn't"


  final = [l for l in lemmed if not l in stop]


  tag_noun = [l for l, pos in pos_tag(final) if pos.startswith("N")]
  tag_verb = [l for l, pos in pos_tag(final) if pos.startswith("V")]
  tag_adj = [l for l, pos in pos_tag(final) if pos.startswith("J")]


  words = tag_noun + tag_verb + tag_adj
  final =  ' '.join([str(w) for w in words])
```

```
    return final


df['text'] = df['text'].apply(LiteCleaning)


EmoWords = df.text
EmoWords = ' '.join([str(E) for E in EmoWords])

shooters_only_plot.text = shooters_only_plot.text.apply(LiteCleaning)
bloggers_only_plot.text = bloggers_only_plot.text.apply(LiteCleaning)

EmoWordsShooters = shooters_only_plot.text
EmoWordsBloggers = bloggers_only_plot.text

EmoWordsShooters = ' '.join([str(E) for E in EmoWordsShooters])
EmoWordsBloggers = ' '.join([str(E) for E in EmoWordsBloggers])

ShootersInput = NRCLex(EmoWordsShooters)
BloggersInput = NRCLex(EmoWordsBloggers)

ShootersEscore = ShootersInput.raw_emotion_scores
BloggersEScore = BloggersInput.raw_emotion_scores

EmoShootersDF = pd.DataFrame.from_dict(ShootersEscore, orient = 'index')
EmoShootersDF = EmoShootersDF.reset_index()
EmoShootersDF = EmoShootersDF.rename(columns = {'index':'Classification', 0: "Count"})
```

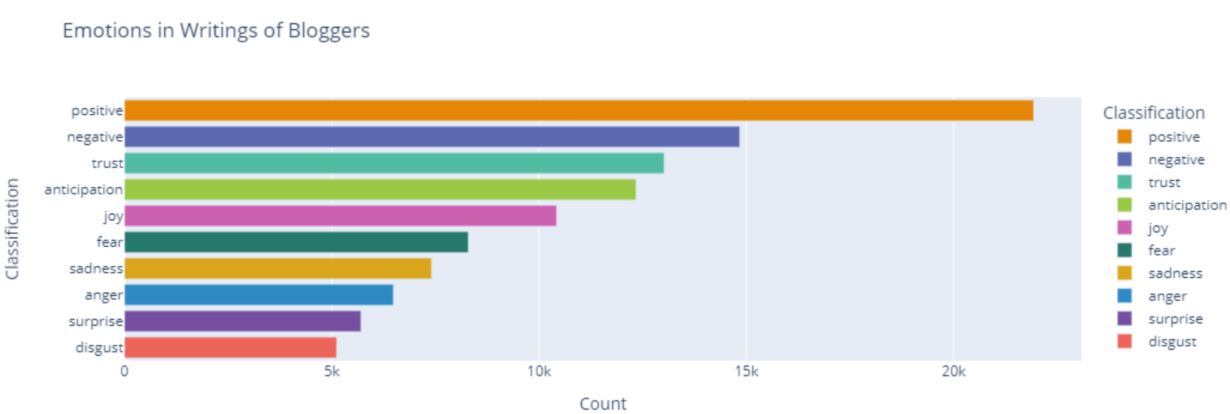
```
EmoShootersDF = EmoShootersDF.sort_values(by = ['Count'], ascending = False)


EmoBloggersDF = pd.DataFrame.from_dict(BloggersEScore, orient = 'index')

EmoBloggersDF = EmoBloggersDF.reset_index()

EmoBloggersDF = EmoBloggersDF.rename(columns = {"index":"Classification", 0:"Count"})

EmoBloggersDF = EmoBloggersDF.sort_values(by = ['Count'], ascending = False)



word = EmoWords

emoScore = text_input.raw_emotion_scores

emotion_df = pd.DataFrame.from_dict(emoScore, orient = 'index')

emotion_df = emotion_df.reset_index()

emotion_df = emotion_df.rename(columns={'index' : "Classification", 0: "Count"})

emotion_df = emotion_df.sort_values(by = ["Count"], ascending = False)


Emofig = px.bar(emotion_df, x = "Count", y = "Classification", color = "Classification", title = "Emotions Detected with Bloggers and Shooters",

        orientation = "h", width = 800, height = 400)

Emofig.show()
```



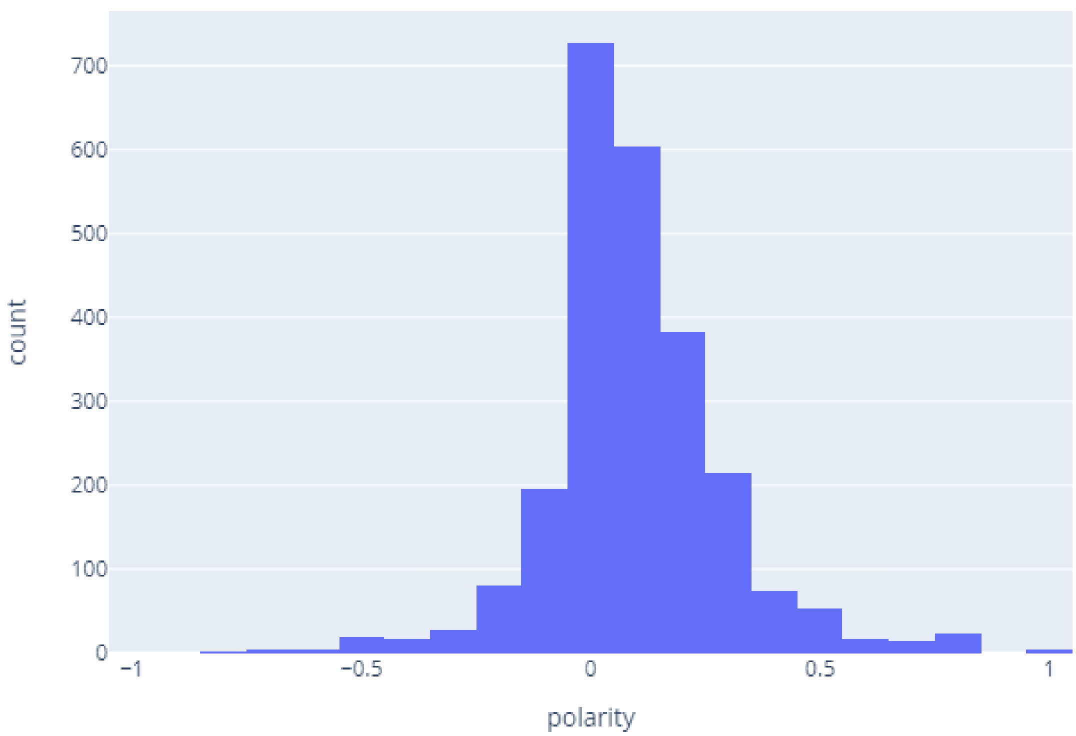

```
EmofigShooters = px.bar(EmoShootersDF, x = "Count", y = "Classification", color = "Classification",
                color_discrete_sequence=px.colors.qualitative.Vivid,
                title = "Emotions in Writings of School Shooters",
        orientation = "h", width = 800, height = 400)
EmofigShooters.show()
```




```
EmofigBloggers = px.bar(EmoBloggersDF, x = "Count", y = "Classification", color = "Classification",
                color_discrete_sequence=px.colors.qualitative.Vivid,
                title = "Emotions in Writings of Bloggers", orientation = "h", width = 800, height = 400)
EmofigBloggers.show()
```



```
df['polarity'] = df['text'].map(lambda text: TextBlob(text).sentiment.polarity)
df['review_len'] = df['text'].astype(str).apply(len)
df['word_count'] = df['text'].apply(lambda x: len(str(x).split()))


cl = df.loc[df.polarity <= 0, ['text']].sample(5).values
for c in cl:
   print(c[0])


f1 = px.histogram(df, x = 'polarity', nbins = 35, width = 800, height = 500)
f1.show()
```

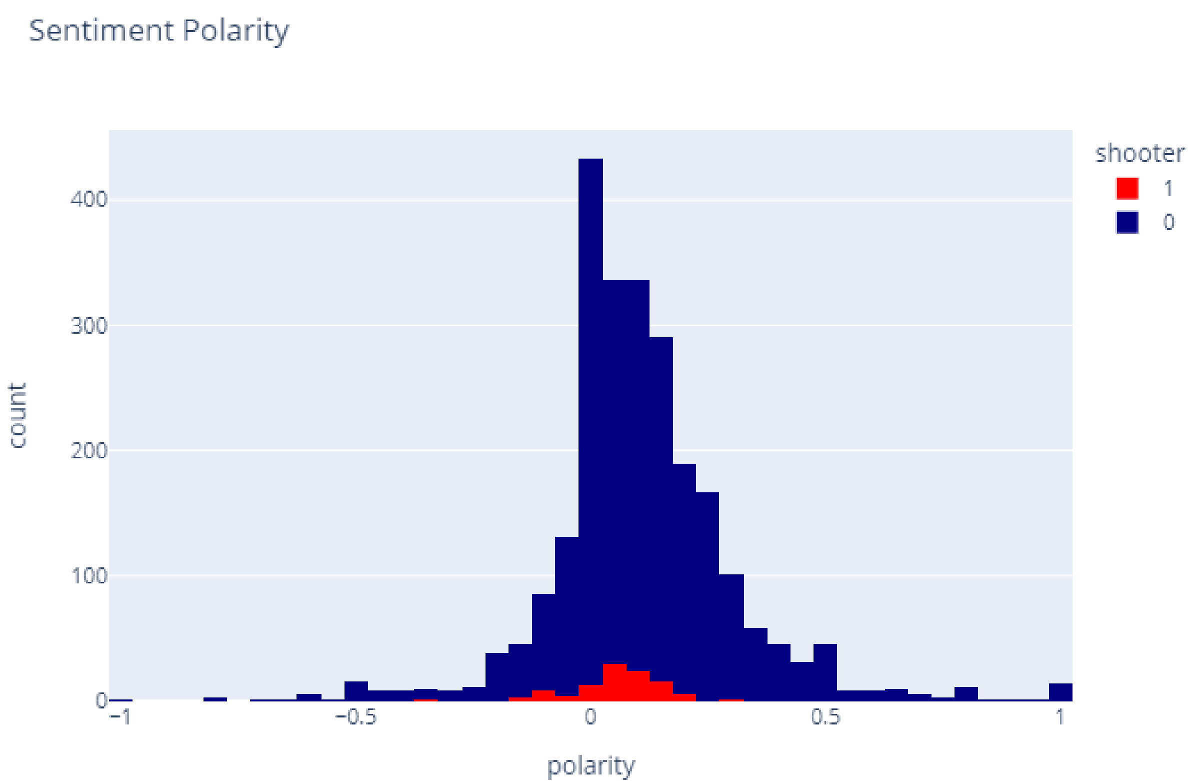


Sentiment polarity with the bloggers compared to the shooters

f2 = px.histogram(df, x = 'polarity', color = 'shooter', nbins = 100, color_discrete_sequence = ['red', 'navy'], title = "Sentiment Polarity",

width = 800, height = 500)

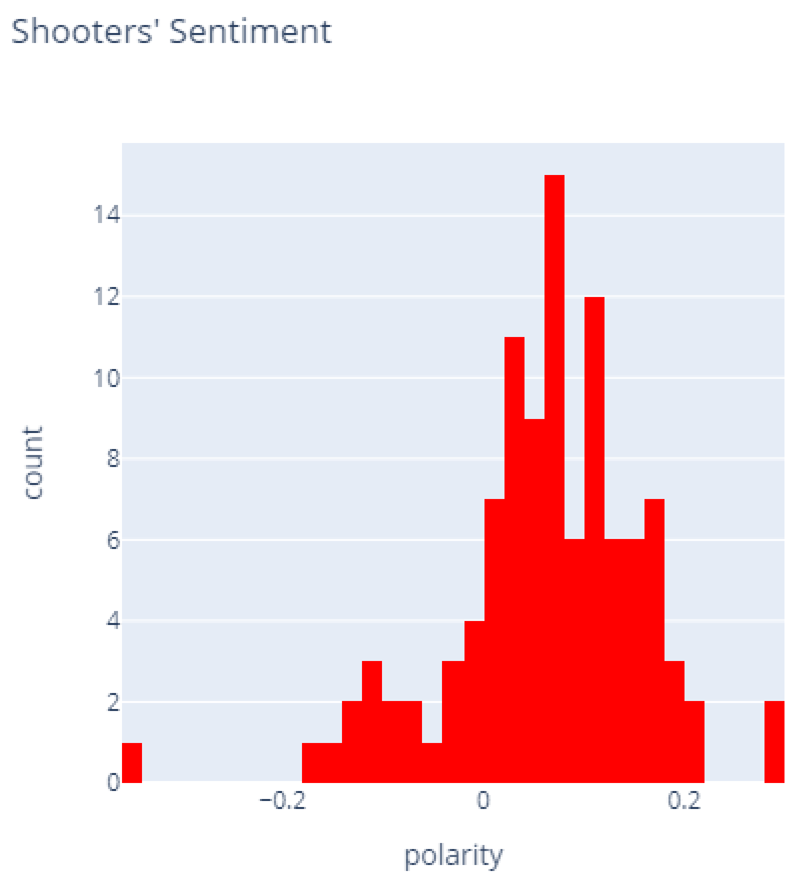
f2.show()



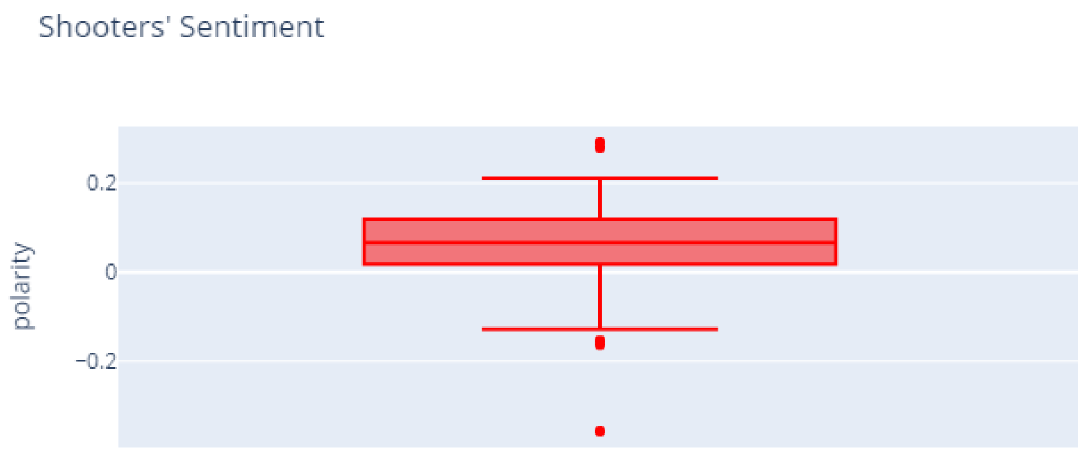
f3 = px.histogram(shooters_only_plot, x = "polarity", nbins = 50, color_discrete_sequence = ['red'],

title = "Shooters' Sentiment", width = 800, height = 500)

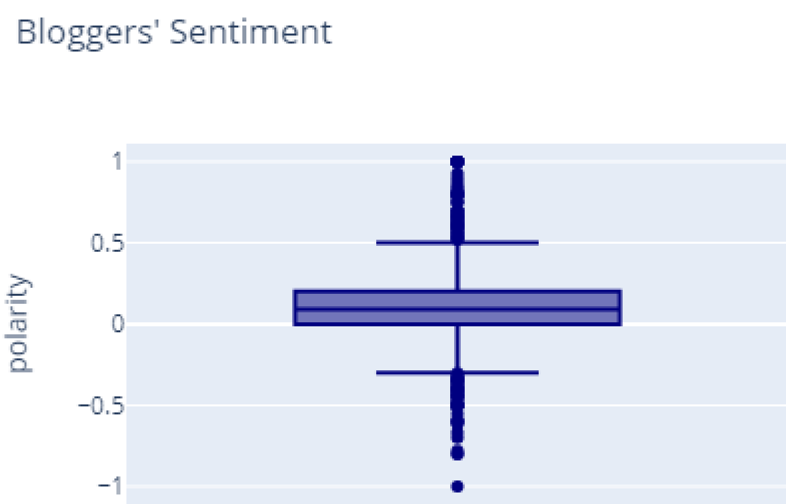
f3.show()




```
bp = px.box(shooters_only_plot, y = "polarity",
          color_discrete_sequence = ['red'],
         title = "Shooters' Sentiment")
bp.show()
```




```
bpb = px.box(bloggers_only_plot, y = "polarity",
          color_discrete_sequence = ['navy'],
         title = "Bloggers' Sentiment")
```



```
bpb.show()
```