Identifying Satire Newspaper Articles

Felix Yang, Chris He, Derek Leung, Ethan Lee

I. Abstract:

The Internet and rise in technology has revolutionized the way people can consume news and information, especially in an era where there is an abundance of information [1]. However, it has also created new challenges related to the truthfulness of the information being shared. Satirical news websites, in particular, are a major concern as they can be mistaken for genuine news sources by readers. While satirical news can be entertaining and thought-provoking, it can also have unintended consequences when readers take it at face value. With the ease of outputting and sharing information due to social media, these readers can easily propagate the spread of such misinformation. As such, understanding when a news article is satirical is imperative towards a better understanding of the real world and the spread of true information. This paper aims to analyze the methods of satirical news identification. Focusing on classifying articles as satirical or non-satirical, the paper focuses on the vocabulary used in the articles, which are from a dataset of scraped articles of top reddit posts from r/nottheonion and r/TheOnion of 2022. The models compared include Perceptrons, Support Vector Machines, Decision Trees, and Naive Bayes. The performance metric of accuracy was utilized to measure the performance of the models as it is imperative to have a firm understanding of what type of article it is, whether satirical or not. The features mainly tested included various word count and semantic values. This paper provides valuable insights into the use of machine learning models for the classification of satirical news articles and explores the use and development of more accurate and effective models for this task.

II. Introduction/Background:

Satires are pieces of writing that often use humor, irony, or exaggeration to expose and criticize contemporary politics or other topical issues. For this project, we will be working to leverage machine learning models to classify a given news article as either satirical or not satirical. We found a dataset with links to various news articles and scraped these sites for article text. Each of the articles is mapped to either “TheOnion” or “NotTheOnion” in the dataset specifying whether or not the article was written by The Onion, an American satirical news website [2]. The Onion's articles are written in a style that is similar to that of traditional news outlets, however, the website's articles are always clearly labeled as satire. The Onion also includes a disclaimer on its website that states that its articles are "not meant to be taken seriously." As an example of a satirical headline, The Onion recently published an article claiming “Sam Bankman-Fried Sobs After Accidentally Dropping Last Crypto Down Sewer Grate.”

The significance of being able to classify satire vs non satire news articles is that it allows people to understand the difference between what is real and what is not. Satire is a form of humor that uses exaggeration or irony to criticize something, while news is supposed to be factual information. It is important to be able to distinguish between the two because satire can be easily misinterpreted as news, and vice versa. This can lead to people believing things that are not true, or to people taking satire seriously. In addition to aiding in the detection of fake news and misinformation, the broader ability for models to be able to understand language style and tone has extreme significance when it comes to computers being able to better communicate with humans. Advancements in voice assistants, chatbots, language translation, and information retrieval are all dependent on the ability of models to understand free text.

When it comes to text classification tasks and understanding writing styles, there are a variety of machine learning algorithms that can be used. Naive bayes, support vector machines, decision trees, and perceptrons are all very capable. However, deep learning models such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) have shown to be very impressive especially for tasks involving larger portions of free text [3], thriving even more with extremely large datasets. Massive neural networks are certainly the state of the art when it comes to chatbots, as seen by ChatGPT, offering a new level of capability in terms of understanding text and answering questions. As a means of examining past attempts to conduct text classification, we looked at a published article which compared multiple algorithms for a similar text classification case working with a large dataset of Stack Overflow posts [4]. Their Naive Bayes Classifier achieved 74% accuracy, Linear Support Vector Machine achieved 79% accuracy, and their Deep Learning Classifier achieved an 80% accuracy. Based on these results, support vector machines and deep learning performed best for this text classification case. This supports the matter that Linear Support Vector Machines are widely regarded as one of the best text classification algorithms. It also backs up the fact that Deep Learning is very impressive and has the ability to achieve state of the art results when handling complex text classification tasks.

The Data

The data was processed using the pushshift.io API [5] in order to scrape reddit posts and their content. Specifically, we took a look at the top 1000 posts of r/nottheonion (a subreddit dedicated to true news articles that seem fake) and the top 1000 posts of r/TheOnion (a subreddit dedicated to satirical fake news articles from theonion.com). We took the article title, the url, and the subreddit for each post and used the Newspaper3k api library in order to scrape the article's text from each of the urls. The text was preprocessed to strip the text of stopwords and punctuation as well as lemmatizing the words to strip each word down to its base. Vocabulary is the main focus of our investigation as we also removed various proper nouns that were only relevant to their own news article. After removing other bare articles that were either missing its text or of a bare quantity of information, there are the following: 864 Onion articles and 696 Non-Onion articles. Each article contains a 'counter' column as a Counter object of each article's words and their frequency for future use in model feature selection.

III. Data Analysis:

Satire vs Non-satire Word Popularity: To start our exploration of the data we decided we wanted to look into the most common words for the Onion and Non-Onion articles and see if there was any commonality. We decided to visualize this by using a sankey diagram showing the top 10 most popular words for each of the Onion and Non-Onion articles, and showing how they flowed from both categories. In addition, bar charts show the number of occurrences of the top 30 words on both categories of articles (Figure 1). Upon looking at the diagrams, we can see the frequency to which words are associated with either Onion or Non-Onion articles. Interestingly, we observe that certain words such as "said, people, and year" are much more commonly used by Non-Onion articles as seen by the greater majority of occurrences flowing to the Non-onion category. In addition, although "said" was the most popular word in both categories, it was over twice as popular as the next most commonly used word in the Non-Onion articles.

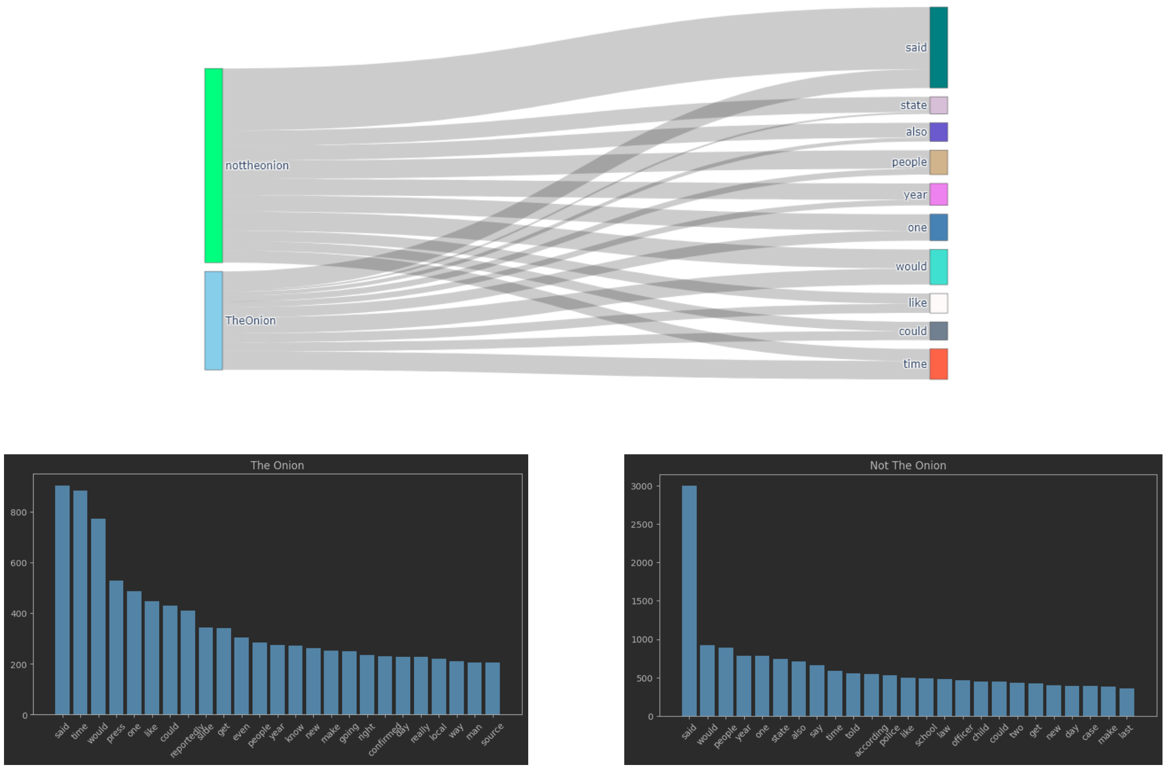

Figure 1. Sankey and Bar Charts of Word Popularity

As another means of describing the word popularity of the Onion and Non-Onion article data, we decided to create word clouds with size depicting the popularity of that word. As observed in the word clouds below, the word "said" and "would" are both very popular among both categories. When looking at differences between the clouds, the words "people, state, and year" are noticeably more prevalent in Non-Onion articles. Meanwhile, words "press, could, and reportedly" appear more frequently in Onion articles. Word popularity could certainly prove to be a useful feature in future model implementation, seeing as certain words are more closely associated with either satire or non-satire writing styles.

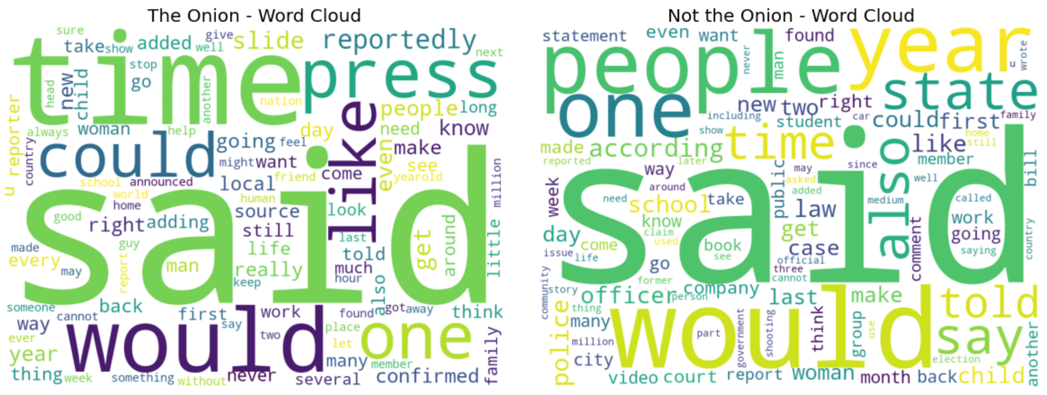

Figure 2. WordClouds

Sentiment Analysis: As part of our examination of the data, we decided to investigate the sentiment of both Onion and Non-Onion articles. To do this, we wanted to first visualize the distribution of articles' sentiment from -1 (most negative) to +1 (most positive). We utilized the TextBlob natural language processing library to rank every article's sentiment and then created two histograms, one each for the Onion and Non-Onion articles. The histograms are displayed below.

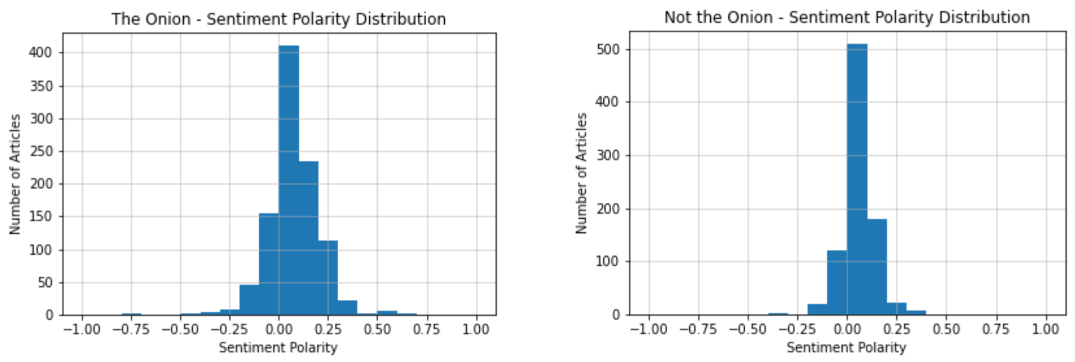

Figure 3. Sentiment Histograms

From observing the distributions, it appears that both Onion and Non-Onion articles centralize around a slightly positive sentiment between 0 and 0.1. However, it is interesting to note that the Onion articles have a wider spread with a greater number of articles deviating into more negative and more positive sentiments. Seeing as the Onion articles are satires, a form of writing which often comments on social or political issues in a way that is deliberately exaggerated or absurd, it is not surprising to see that more strongly positive or negative sentiments were observed from that dataset. Meanwhile the Non-Onion articles had a more narrow distribution highly concentrated in the range of -0.1 to 0.2.

IV. Methods:

We implemented a variety of ML models for the purpose of comparing their approaches and overall performance in classifying Onion and Non-Onion articles. We decided to implement the following 4 models and will elaborate on their individual feature selection, rational, tuning methods, and cross-validation approach.

Perceptron: The perceptron model was implemented similarly to class without the use of scikit-learn libraries, but unlike class the model needed to be implemented for the purpose of text classification. The free text Onion and Non-Onion articles do not have predefined numeric attributes to use as features, so they needed to be extracted from the text. Based on research findings, we decided to move forward by using the 'bag of words' feature extraction approach [6] as a means of transforming free text into a useful numeric form. As described in the article, the bag of words approach defines a set of unique words within the dataset and then a vector of counts is created to represent each data point's features. It is a simple and efficient NLP approach to feature extraction and can result in a very large number of features with feature vectors being very sparse containing lots of zeros. This was certainly seen when expanding the 'counter' column, which holds each article's word counts into separate columns for every unique word, into new feature columns with every unique word having its own column. There were thousands of unique words in the dataset resulting in thousands of feature columns. To avoid including irrelevant features, we selected the most common 500 words and decided to use these as our features for the model. Next, the perceptron model was implemented, adjusting the weights with an activation function for each feature to reduce the mean percentage error. This process was completed over 5 fold cross validation with 500 epochs per fold, and the average accuracy of the 5 models was 85.38%. Below is a visualization of the Accuracy vs Epoch for each of the 5 folds.

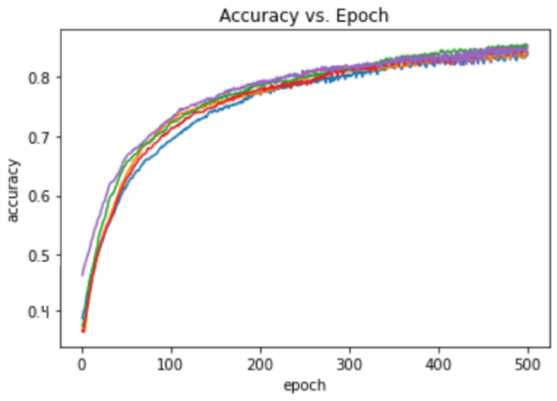

Figure 4. Accuracy vs. Epoch

Support vector machines: As a discriminant technique, SVMs construct their solution from the training input by avoiding outer probabilities in determining separable classes. They optimize a solution analytically and return a unique hyperplane parameter to classify, so it is constructed quicker on good representative data [7]. Especially good in binary classification due to their robustness, the model was implemented using sklearn's `SVC`. The features are defined on the cleaned text data vectorized with sklearn's `TF-IDF `(TF) and `CountVectorizer` (CV) as vocabulary count features throughout the texts. The performance accuracy of the model was that of 93.08% when vectorized with TF (figure 5) and that of 90.13% when vectorized with CV.

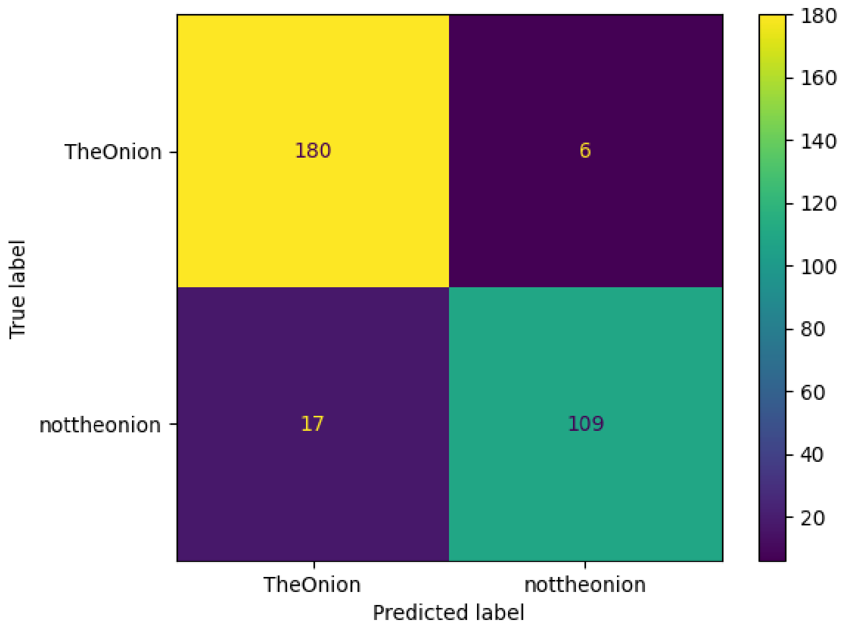

Figure 5. Confusion matrix of SVMs

Decision trees: The decision tree model was implemented using sklearn's `DecisionTreeClassifier`. We defined a couple hyperparameters to tune on using sklearn's `GridSearchCV`. As training data, we used the cleaned version of the articles, which was represented in a counter as words and the number of occurrences that they appeared, with stopwords filtered out. We did a test train split of 70/30, and the results of the test are shown below. The function's input was a string of text that it then cleaned and passed to the model. The final performance of the model was 85.2%. We believe that this can be improved with the addition of more hyperparameters to tune on.

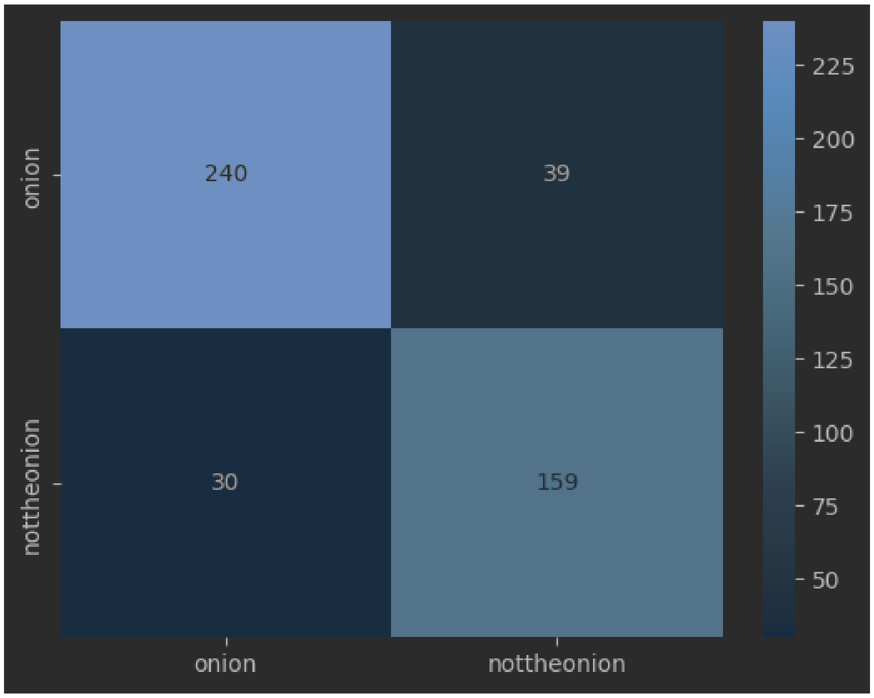

Figure 6. Confusion matrix of decision trees

Naive bayes: The Naive Bayes model was implemented using sklearn's 'MultinomialNB', which was trained on a subset of 1000 out of the 1688 total articles' texts. This classifier applies Naive Bayes with independence between the features, and is a model known to be suitable for classification with discrete features. We then transformed the data using 'CountVectorizer' to link the index value of each word in the vocab to its total frequency, and 'TfidfTransformer' to avoid discrepancies in average count values between larger and smaller documents. This process achieved an accuracy of roughly 95%.

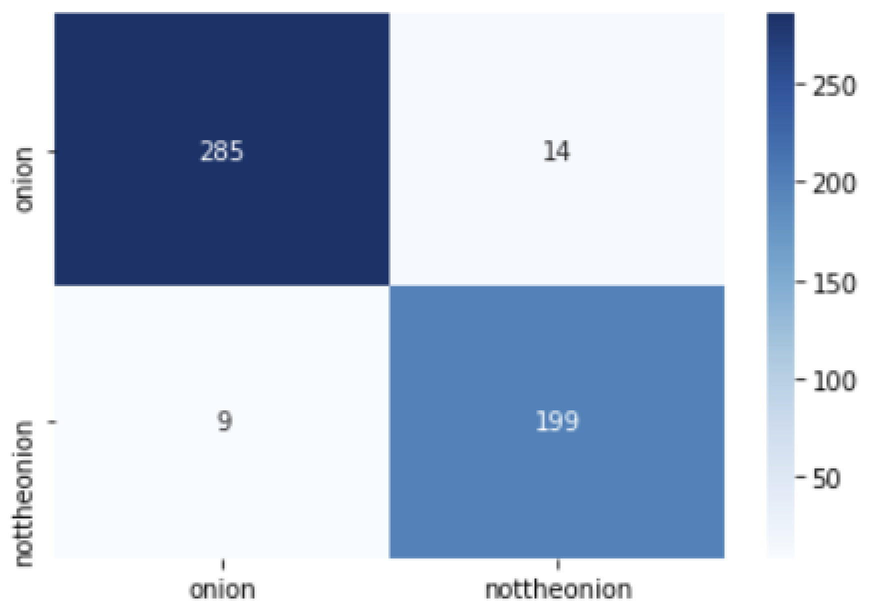

Figure 7. Confusion matrix for Naive Bayes

V. Analysis:

After completing the implementation of our various machine learning algorithms, we can now compile results and compare their performance. To summarize, the perceptron model had an average accuracy of 85.38%, support machine vectors an accuracy of 93.08%, the decision tree model an accuracy of 85.25%, and naive bayes an accuracy of 95.46%. While perceptrons and decision trees achieved a substantial accuracy at about 85%, the support machine vectors and naive bayes clearly performed the best at 93.08% and 95.46% respectively.

These findings support our group's previous research that support machine vectors are widely regarded as one of the best text classification algorithms. As we discovered in our implementation of support vector machines, they are very well suited to text classification particularly in part because they work to achieve an overall optimization as opposed to a greedy search methodology which takes only the optimal decision at given points in time regardless of the larger optimization picture. Support machine vectors are also ideal for text classification in that they can effectively handle the thousands of features that come with free text data such as the news articles we are dealing with. This is very different from the perceptron model, which originally contained over 20,000 unique word features and had to be cut down to the top 500 most frequent words for algorithm efficiency. It took over 20 minutes to run perceptrons over a 5-fold cross validation with 500 epochs each, as opposed to the support machine vectors which ran very quickly.

Additionally, the naive bayes model slightly outperformed our support machine vectors model by a few percent. Naive bayes is one of the most popular algorithms to use for text classification and it relies on the probability of an object to make predictions. For our case in particular, it relied on the probability that each word in the article would be in either an Onion article or a Non-Onion article. This clearly worked very successfully in classifying seeing as it was our highest performing model at 95.46%. We can assume that based on this high accuracy certain words are very highly probable to show up in a satire and certain words are very probable to show up in a non-satire. This goes along with our observations in our exploratory data analysis where we saw in the sankey diagram that specific words flowed much more to the Onion category than Non-Onion category, and vice versa. Since satirical work is often very exaggerated and bold, it makes sense that specific, possibly more flashy language, would appear more frequently in The Onion and allow naive bayes to be an effective algorithm for our text classification.

VI. Conclusions:

Throughout this project, we researched and implemented 4 methods of text classification (perceptrons, support machine vectors, decision trees, naive bayes) in order to classify news articles as either a satire piece or a non-satire piece. We used data on news articles, each of which was tagged as either "TheOnion" specifying it was a satire piece, or "NotTheOnion" specifying a non satirical article. The overall accuracies of each model in ascending order were 85.38% for perceptrons, 85.25% for decision trees, 93.08% for support machine vectors, and 95.46% naive bayes. The support machine vectors and naive bayes clearly outperformed our other two methods. Both proved to be very successful at text classification, with support machine vectors being very effective at overall optimization among a high number of dimensions that come with news article text, and naive bayes identifying and utilizing the fact that there were key words that showed up much more commonly in satire and non-satire articles. In all, our results largely backed up our research on text classification algorithms.

VII. Author Contributions:

Each member contributed to major parts of both the code and the paper.
Perceptrons - Ethan
Support machine vectors - Felix
Decision trees - Derek
Naive Bayes - Chris
Paper write up - all members contributed

VIII. References:

[1] van der Linden, S. Misinformation: susceptibility, spread, and interventions to immunize the public. Nat Med 28, 460–467 (2022). https://doi.org/10.1038/s41591-022-01713-6

[2] theonion.com

[3] Li, Hang. "Deep Learning for Natural Language Processing: Advantages and Challenges." Academic.oup.com, https://academic.oup.com/nsr/article/5/1/24/4107792.

[4] Li, Susan. "Multi-Class Text Classification Model Comparison and Selection." *Medium*, Towards Data Science, 6 Dec. 2018, https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568.

[5] https://github.com/pushshift/api

[6] Kaul, Shachi. "Text Feature Extraction (1/3): Bag of Words Model." Medium, Geek Culture, 28 June 2021, https://medium.com/geekculture/text-feature-extraction-1-3-bag-of-words-model-649dbeeade79.

[7] Awad, M., Khanna, R., Awad, M., & Khanna, R. (2015). Support vector machines for classification. Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers, 39-66.