Sentiment Analysis of News Articles

Dhanush Dharmaretnam, Madhav Malhotra, Jasmeet Singh, Evan Wilde
Department of Computer Science, University of Victoria, Canada.
Email: dhanushd,madhavmalhotra,jasmeets,etcwilde @uvic.ca

***Abstract*—Violence in the media is becoming more prolific than ever. Violent content in news articles is detrimental to the psychological growth of minors and is not appropriate for their consumption. With the availability of online material, minors have access to violent materials like never before.**

In this paper, we build and test the capabilities of three Naive Bayes classifiers, Linear SVM classifier, decision trees, random forests, adaboost with logistic regression, and extra tree classifier to perform sentiment analysis to determine which articles are appropriate and inappropriate for minors. We work with a subset of the data from the GDELT project, which performs analysis on news articles to determine the events and actions taking place in the article. GDELT breaks the articles into the 20 CAMEO codes, and further into verbal cooperation and conflict and material cooperation and conflict that we use for classifying whether or not something is appropriate or inappropriate.

I. Introduction

The subject of sentiment analysis is a widely studied topic. Generally, sentiment analysis works well with short clips of text, as these are capable of only containing a single sentiment. Twitter [1] provides an environment that is perfect for sentiment analysis tasks, where users are forced to write each tweet within 140 characters or less. It becomes much more difficult to classify longer texts, as the sentiment may change over the piece of text. In this case, it would be difficult for a human to classify the text into a single sentiment as well.

News articles are longer pieces of text, possibly containing multiple sentiments, making it more difficult to perform sentiment analysis; however, news articles are in a restricted domain. A news article focuses on a single topic, which we may be able to leverage in determining whether that topic is appropriate for consumption by a minor.

With the increase in problematic media, a Global Database of Event Language and Tone (GDELT) was created as a means of aggregating the broadcast of information identifying the people, locations, organizations, themes, sources, emotions, quotes, and images involved [2]. GDELT categorizes news articles into 20 CAMEO [3] codes;

- Make Public Statement
- Appeal
- Express Intent to Cooperate
- Consult
- Engage in Diplomatic Cooperation
- Engage in Material Cooperation
- Provide Aid
- Yield
- Investigate
- Demand
- Disapprove
- Reject
- Threaten
- Protest
- Exhibit Military Posture
- Reduce Relations
- Coerce
- Assault
- Fight
- Engage in Unconventional Mass Violence

GDELT further breaks the 20 CAMEO codes into four subcategories;

- Verbal Cooperation
- Material Cooperation
- Verbal Conflict
- Material Conflict

The twenty CAMEO codes are mapped into the four categories of cooperation and conflict, and so an article that is mapped into one of the twenty can be mapped directly into one of the four.

We use the GDELT [2] database as our source of data, and as our source for the ground truth. We make a minor modification to their data, combining the verbal and material cooperation into the cooperation category, and the verbal and material conflict into a conflict category. With this, we have access to newly classified and processed articles every 15 minutes from sources all over the world.

II. Related Work

Sentiment analysis on text from various sources can be used for many purposes including; automatically rating movies, determining the outcome of an election, the projection of a company stock prices [4], determining the location, time, and culprit behind a crime before it happens [5]. Sentiment analysis is good for collection opinions [6] and determining how people may respond to a situation.

Sentiment analysis of news articles has been done before with the intent of determining the behaviour of stock prices [4]. While this doesn't directly correspond with gathering information about articles that are appropriate and inappropriate for minors, it does mean that it is possible to classify larger textual articles based on sentiment.

Sentiment analysis of news articles has also been conducted by social scientists to investigate topics related to the public perception of race, gender, and religion based on the biases of news articles [7]. Other uses of sentiment analysis of news

articles includes ranking the strength and legacy of presidents [8].

While we are working on this, it is important to keep in consideration the implications of censorship. There is much debate on whether a company, government, or even family members should have power over what content an individual has access to, as these will characterize the views of that individual.

III. Methodologies

In this section, we outline how we conducted our experiments. This includes information on how we collected the data, vectorized the text, built the classifiers, and conducted the tests.

A. Data Collection

The raw data is available from the GDELT project via file transfer[1]. This file contains links to the csv files with the categorized articles. It has a link to a csv file for every day back to April 1st 2013, then every month from March 2013 to January 2006, and then yearly from 2005 back to 1979. The csv files contain information on which CAMEO code an article belongs to, the people or entities performing the action, the actions occurring, and a link to the article itself. While this information could be useful, it does not include the further breakdown into cooperation and conflict of each article. For that, users must request the data via email[2].

We queried GDELT for the articles between January 1st 2016 to October 30th 2016, resulting in 30,259 articles total. We extracted the link and class from the csv that was sent to us. We took the class and converted it from the four super classes to two classes where one represented conflict and the other represented cooperation. We re-labelled data with the label “verbal conflict” or data with the “material conflict” as “conflict”, and “verbal cooperation” or “material cooperation” as “cooperation”. Internally, we represented conflict as being 0, and cooperation as 1. We took the list of articles and randomly split them into a 20%, 80% split, with 6,053 articles going to the test set and 24,206 articles going to the training and validation set.

The GDELT dataset only provides a link to the article, not the full text used to classify the article. We built a simple scraper to download the article from the web and cache it locally. Working with the web data proved to be difficult; some sites were taking steps to combat web scraping. For most of these, we were able to get around their techniques by setting our user agent to one that would appear on a normal web browser. Specifically, we provided a user agent corresponding to a Google Galaxy Nexus Build, running Android 4.0.4 using Chrome version 18 as a web browser. We chose a mobile platform, as most mobile devices limit the amount of javascript, and so mobile versions of the site will have less javascript, which will affect our ability to classify the text. Other sites embedded the text of the article in javascript in either the head or body of the html document, only to be rendered by a javascript engine. Most techniques for extracting the text of the webpage failed due to these steps taken to break web scraping. Instead of removing these articles, we vectorized and trained on the entire html file. Luckily, the insertion of most advertisements is via javascript and is not present in the initial html document sent to our client, so our results are not as skewed by the contents of advertisements.

B. Text Vectorization

The classifiers we worked with cannot directly take text as input. Instead, the text must be broken down into vectors of words. There are various techniques for vectorizing text information.

There are three main text vectorizers provided with scikit-learn, the count vectorizer, the hash vectorizer, and the term frequency-inverse document frequency (TF-IDF) vectorizer.

We used the TF-IDF vectorizer, as it selects the most important terms from a document rather than simply by the frequency of a word. We worked with the stop words for English text from scikit learn[3]. If we did not include stop words, the count vectorizers would recognize words like "and", "the", and "a" to be of high importance; however, using the inverse document frequency will likely remove these words as they will show up in almost every document and are therefore not important for distinguishing one file from another. Similarly, the tags used in the html will also be removed as they show up in every document. From the TF-IDF vectorizer, we only keep the top 1500 words for performing the classification. With more words than this, the matrices for testing become too sparse for good classification.

C. Classification

We worked with nine classifiers from the scikit learn library to see which classifiers behaved best for the purpose of classifying text. These classifiers include the Bernoulli Naive Bayes, Multinomial Naive Bayes, Gaussian Naive Bayes, Linear SVM classifier, decision trees, random forests, extra trees, and adaboost with logistic regression.

Training each classifier is simple as each can take the same text vector as input and classify on that input. While training, we construct two files; one is the vocabulary file, which are the words we classify on. The second file is a pickle of the classifier itself. The vocabulary file allows us to ensure that the same vocabulary is used for each classifier during testing and training. It also improves the performance of training since much of the time is spent constructing the vocabulary.

D. Tuning

Each classifier has hyperperameters that need tuning in order to get the best behaviour. Over-tuning these parameters will result in overfitting, which we would like to avoid.

Naive Bayes: The Naive Bayes classifiers tuned the α parameter. We worked with $\alpha = 1 \times 10^{-}6$, $1 \times 10^{-}5$, 0.0001,

[1]http://data.gdeltproject.org/events/index.html

[2]http://analysis.gdeltproject.org/module-event-exporter.html

[3]https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/feature_extraction/stop_words.py

0.001, 0.01, 0.05, 0.1, 0.5, 0.9, 1, 5, and 10. We used 10-fold cross validation on the entire training set to see where overfitting began and to see where the point of diminishing returns began. Tuning α applied to Bernoulli and Multinomial naive Bayes, but not to Gaussian Naive Bayes, so we do not tune Gaussian Naive Bayes.

Decision trees: Tuning decision trees involves pruning the tree to ensure that overfitting does not occur. Overfitting of decision trees occurs when the tree is allowed to grow too deep, as each point of data will be assigned to its own leaf, when some data should be clustered in a single leaf. We tuned the maximum tree depth parameter to find the point where the decision tree was well behaved. We tried tree heights starting at 10 and going up to 55 (inclusive) by five. Again, we used 10-fold cross validation on the entire training set to determine the point where overfitting begins.

Extra Trees: We tuned the number of trees in the extra tree forests. We start with 1 and go up to 90(inclusive) trees, increasing by one with each iteration. We train on half of the training set and validate on the other half using 10-fold cross validation.

Random forests: We tuned the random forests on the number of trees to create in the forest. We start with 1 and go up to 86(inclusive) increasing by one with each iteration. We split the training set in half, one half was used for training and the other half for validating. We trained the forest on the training set and ran 10-fold cross validation with the validation set.

Linear SVM: We tuned the linear SVM on the C hyper parameter. This controls how tolerant the SVM is to errors when defining the hyperplane separating the data. We start C at 0.050 and end at 2(inclusive). We increase C by 0.05 with each iteration.

E. Testing

We train the classifiers on all of the training data and save the resulting classifier and vocabulary file. The vocabulary file is important, as vectorizing the input text from the test data will result in different words being associated with each word, which will give us results that do not make sense with that classifier.

With the fully trained classifier, we run each test text through the vectorizer with the pre-defined vocabulary to get the vector representing that article. The classifier classifies it into either appropriate or inappropriate and we compare the result with the ground-truth according to GDELT. We collect the accuracy, recall, precision, and F1 score metrics for each classifier.

IV. Results

In this section, we provide information about the behaviour as we tuned the classifiers and the final results from the testing set.

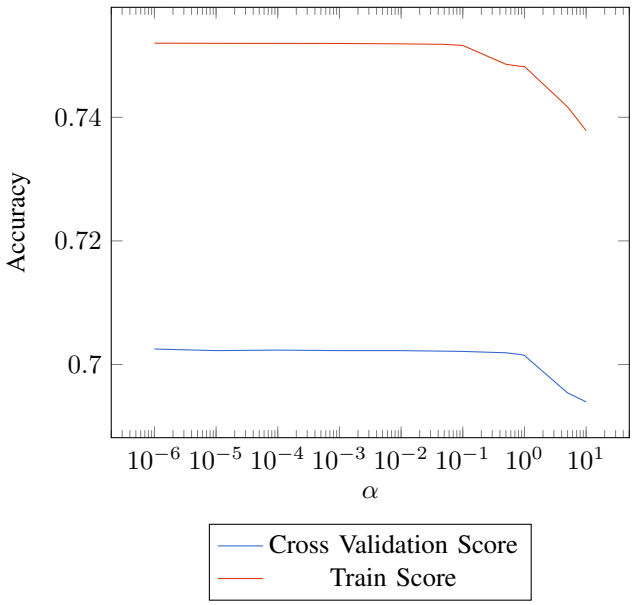

Fig. 1: Accuracy of Bernoulli Bayes over α

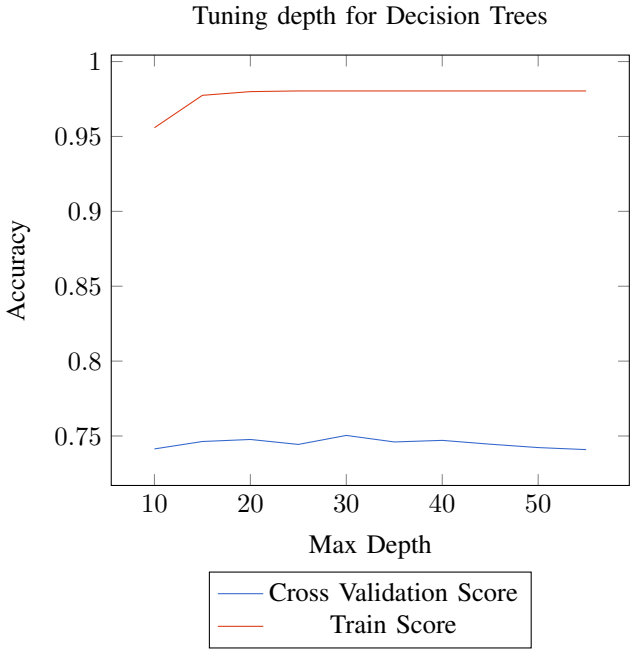

Fig. 2: Behaviour of Decision Trees as Depth is Adjusted

A. Naive Bayes

From figure 1, Bernoulli Bayes works best with a small value for α; however, it still doesn't work as well as some of the other models. We use a logarithmic scale on the x-axis to show the behaviour of the classifier with very small α and relatively large α. The behaviour is consistent until $\alpha = 1$, so we work with the smallest α possible.

B. Decision Trees

From figure 2 we see that the depth of the decision trees does not have a large impact on the behaviour of the classifier

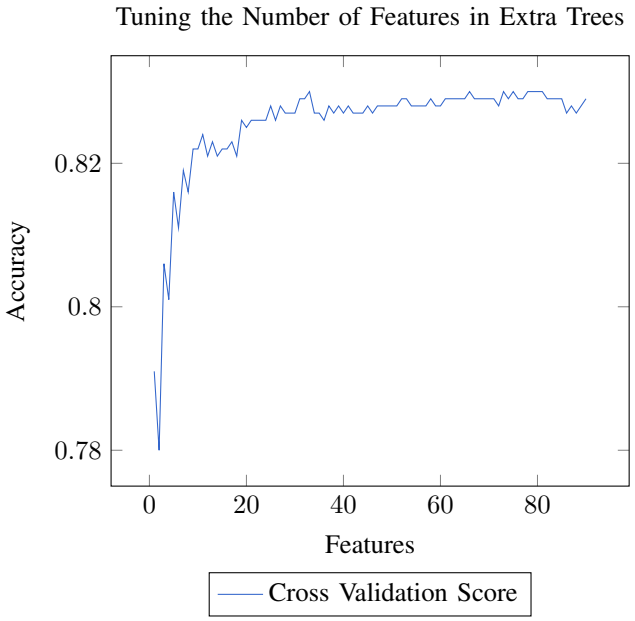

Fig. 3: Behaviour of Extra Trees as the Number of Random Trees is Adjusted

as both the training and validation scores are relatively flat. There is still an improvement in the decision tree from a max depth of 10 to 20, so we work with trees of depth 20.

C. Extra Trees

We tune the number of random trees in the extra tree forest. Like random forest, extra tree constructs trees, but unlike random forests, the splits are completely random. We see the behaviour of the extra tree classifier as we adjust the number of trees in figure 3. We can see that at 33 trees in the forest, we get a maximum accuracy. This is also the point where diminishing returns begins.

D. Random Forests

We tune the number of trees in the random forest. Unlike the extra tree classifier, the trees are not constructed randomly, but instead constructs the trees in the forest using subsets of the data. We can see in figure 4 that the random forests perform quite well. We select the random forest using 50 trees in the forest.

E. Linear SVM Classifier

We tune the tolerance to error in the linear SVM. C starts at a very small number, which means that it is very tolerant to error when defining the support vectors for the hyperplane. With very small C, this line is essentially random. As C increases, the tolerance to error decreases. If C is increased too far, the SVM will overfit and will not perform well in the cross-validation stage. We see in figure 5 that we get good behaviour without diminishing returns in the cross-validation when C is at $C = 0.7$.

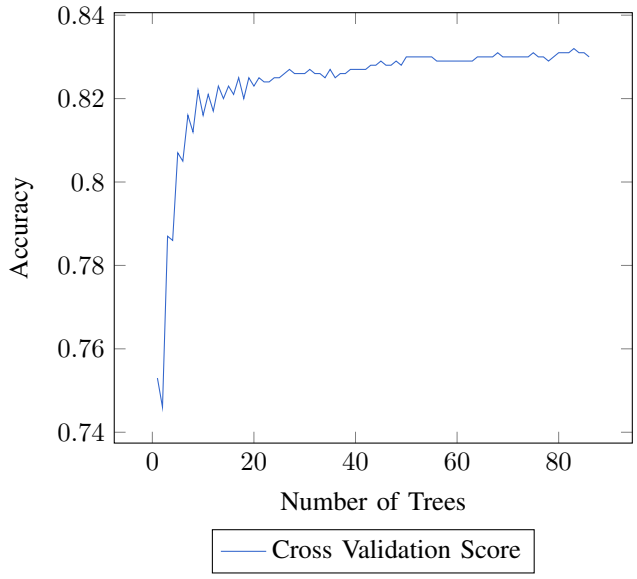

Fig. 4: Random Forest Behaviour as Number of Trees is Adjusted

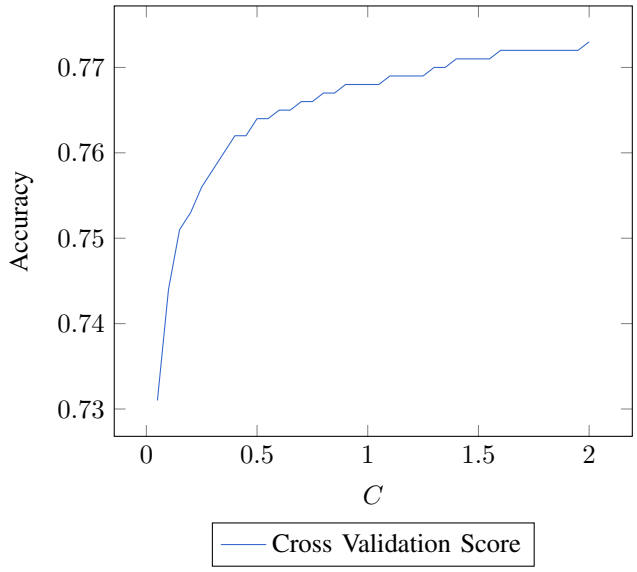

Fig. 5: Linear SVM Classifier Behaviour as C is Adjusted

F. Logistic Regression

We tune the tolerance to error in logistic regression by adjusting the C parameter. When C is very small, the separating hyperplane found by logistic regression will tolerate error more than when it is large. If C is too small, the resulting plane will be meaningless as it is essentially random. If C is too large, the plane will not accept any error and may overfit. We use the plot in figure 6 to help detect when C is being overfit. From the plot, we can see that $C = 0.25$ is where we begin to see diminishing returns.

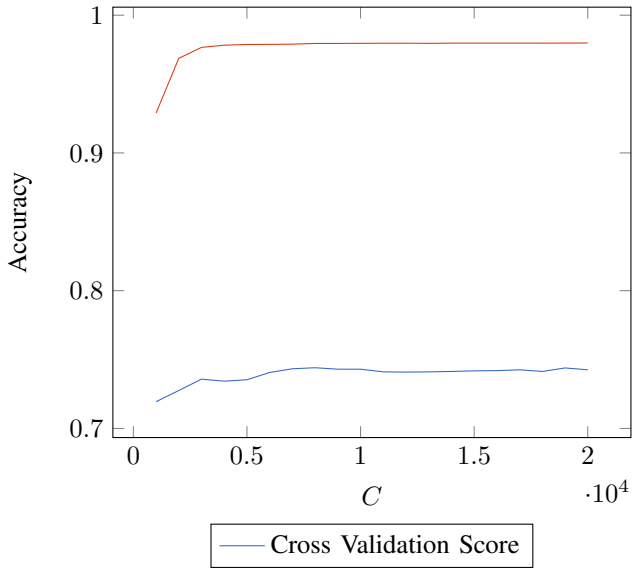

Fig. 6: Logistic Regression Classifier as C is Adjusted

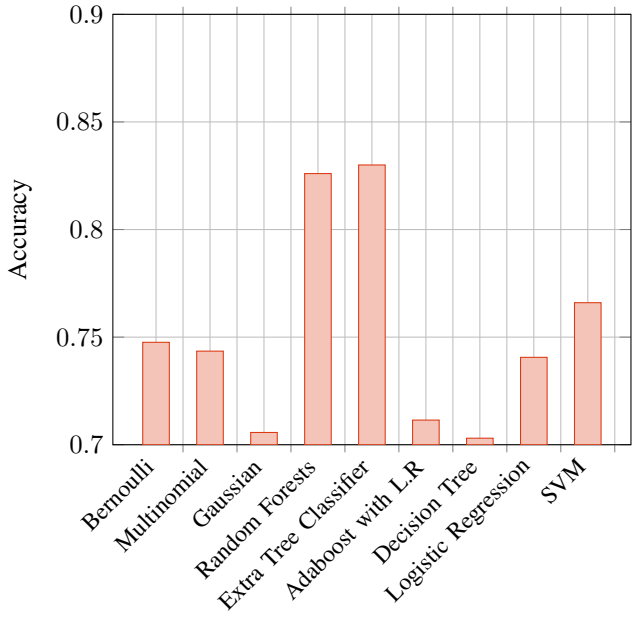

Fig. 7: Cross Validation Accuracy

G. Cross Validation Results

From figure 7, we see the performance of each classifier when running 10-fold cross validation on the classifier, we see that random forests, extra tree classifiers, and SVM perform significantly better than the other classifiers. Bernoulli, Multinomial, and Logistic regression are in the middle ground, and Gaussian, Adaboost with Logistic regression, and decision trees perform the worst on the cross-validation.

H. Test Results

In figure 8, we see that the accuracy for Bernoulli, Multinomial, Gaussian, Adaboost on logistic regression, Decision Trees, logistic regression all saw improvements in performance when run on the test dataset. In comparison, the performance of the random forest, extra tree classifier and SVM remained nearly the same.

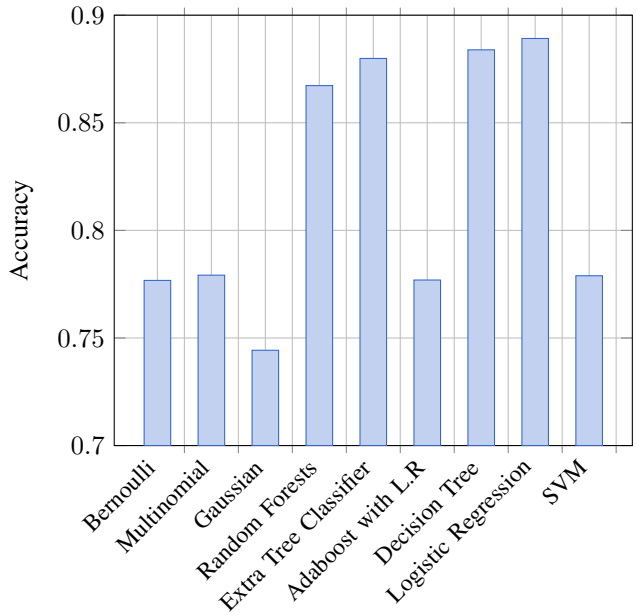

Fig. 8: Accuracy on the Test Data

Figure 9 show the various test metrics. There are minor differences in behaviour, but the best-performing all had nearly indistinguishable differences in the precision and recall, and in extension, the F1 score.

SVM and Adaboost on logistic regression both have better precision, if they report that an article is inappropriate, the article is more likely to be inappropriate, but will not always detect the inappropriate article. The other classifiers are still able to out-perform both in both precision and recall.

V. Discussion & Observations

Here we discuss design decisions and observations we made while working on this project. We discuss the reasoning behind why we chose the TF-IDF vectorizer and not other vectorizers, issues that we ran into with the classifiers from scikit-learn, and issues with the dataset.

We had three options of vectorizers available to us through the scikit-learn library. These included the count vectorizer, hash vectorizer, and TF-IDF vectorizer. The count vectorizer counts the number of occurrences of each word over all documents in the training corpus. This will place more weight on words that occur frequently. Without the use of stop-words, these might include terms like "the", "and", "but", and "I". Hash vectorizers behave similarly to the count vectorizers, but instead of keeping the whole string, the hash vectorizer converts the string into a hash. This improves the memory footprint and improves CPU performance, ultimately reducing vectorization time. For the same reason with the count vectorizer, the hash vectorizer is not what we want. We are

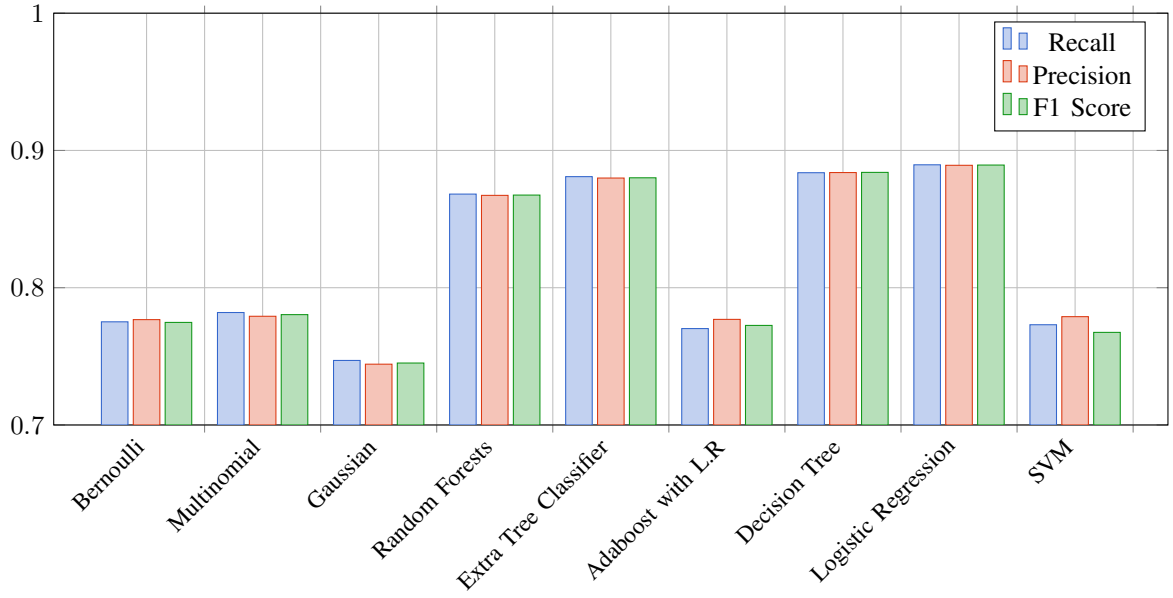

Fig. 9: Testing Metrics for Each Classifier

looking for a way of distinguishing the documents and finding what makes them unique. The count-based vectorizers will provide information on how they are similar, not how they are different. The TF-IDF uses the inverse document frequency to select words that only show up in a few documents from the training corpus, but show up often within those documents.

Initially, we were interested in seeing how the rbf kernel may affect the behaviour of the SVM to see if that would provide better accuracy. Due to the implementation and nature of the data, we were unable to use other kernels than the linear SVM. The implementation in scikit-learn of the general SVM requires that the sparsity of input the matrix be the same. If the SVM is trained with a dense matrix, the prediction input must also be of a dense matrix. While this will work fine for the purposes of training and testing, it does not work for our actual use-case, where we want to classify a single news article. A single news article is unlikely to have all of the top 1500 words most important distinguishing words from the training corpus, and so the vector a single article will result in a sparse matrix, but the entire corpus will have all of the 1500 words from the corpus since that is where the vocabulary is defined. While the testing data does not have the same terms or vocabulary, there are more articles in the test set than a single article, and is more likely to contain more words from those 1500 words.

As we were working with the actual classifiers and testing them with real-world news, we noticed that they mostly tended to be pessimistic, labelling everything as being inappropriate for children. Further inspection showed that the data produced by GDELT was noisy and in some cases made little sense, so even though it looked like our accuracy was fairly decent, the noise in the dataset created a classifier that would classify what was appropriate and inappropriate based on the definition that GDELT was defining in their data. This definition did not match our intuitive sense, and resulted in confusing situations. To test the actual behaviour of the classifiers, we defined a very small training and test set of roughly 20 articles each that we tested and classified on. We found that while the score was much worse, the cross-validation score was less than 50%, the classification of individual articles made far more sense. The drop in accuracy makes sense as there is far less data to train with, and a misclassification will carry a higher weight in testing as there are fewer articles to test on.

Comparing the results of figures 7 and 8, we see interesting behaviour. Maybe most interesting is the improvement in performance of the decision tree. In the cross validation, decision trees have the lowest accuracy, and in the test data, the decision tree had the second highest accuracy. On the other side, the SVM had nearly no change in performance between the cross validation on the training set and the results from the test set. Also interestingly, the logistic regression performed fairly poorly in the cross validation, but it performed the best on the actual test data. This raises some suspicions on the quality of the data and should be investigated further. This may be related to the incredible amount of noise found in the dataset and the type of data. Most grammars can be broken down into a tree-like structure. If instead of a bag of words, we were providing the raw text to the decision trees, it may be able to learn something about the grammar to help it understand the text; however, we are providing the information as vectorized bag of words, so the grammar is not available. This may also be an affect of the different validation techniques used. The cross-validation results from SVM, Random forests, and extra trees used a separate validation set; whereas, the decision trees and logistic regression did not. The Naive Bayes techniques did not provide any further insight or questions.

VI. Threats to Validity

First and foremost, due to some confusion, the data used for training and validation were not originally documented; however, new news is created daily, so a new training and test set was acquired from data that we had not worked with. When this mistake occured, we were trying to classify on the 20 CAMEO codes, so there should be little not no effect on the current classifiers since we are classifying different data into different classes.

The news data used is from January 1st to October 30th of 2016, which is a politically heated time for the United States due to the presidential elections. Given the nature of the candidates, the news surrounding the names of the candidates will bias our results. News with the names of the candidates will be politically charged, and therefore will have the potential, and likely do, skew our results.

The GDELT project uses a machine-based classifier to classify the documents rather than through human classification. Using GDELT as our ground truth will affect our results. We can only be as good as our data, and from manual inspection, the data provided by GDELT contains many errors and lots of noise. The errors added by the GDELT classifier will propagate into our classifier, and so we will at least double their error in the best case.

We do not strip content from web pages. The reasons for this are stated in above in section III-A; however, this still poses a threat to validity. The words contained in a comment of the javascript will influence the classification of the text, even though this text is not visible on the site. Advertisements that are inserted directly into the html without the use of javascript will be visible to the classifier and affect our results as we have no means of stripping advertisements from the articles.

VII. Future Work

We worked with pre-built generalized algorithms provided by the scikit-learn library; however, it would be interesting to see how we could improve the performance by constructing some of these classifiers in a package like Theano or Weka.

As of now, we only classify using a bag-of-words representation. Each word is put into a vectorized vocabulary and those vectors are provided to our classifier. A recurrent neural network may be able to learn more about the text and provide better responses than the bag-of-words representation as it can start to pull more semantic meaning of phrases, sentences, and maybe even paragraphs. Building on this idea, we could use the recurrent neural network to construct a heat-map visualization of news articles to provide insight on what parts have more inappropriate content and which parts have appropriate content.

Finally, providing either an open api or a webpage would make this project more accessible to people who are interested in the censorship of information.

VIII. Conclusion

Our goals were to perform sentiment analysis on news articles to determine which articles are appropriate for children and which are inappropriate. To do this, we worked with nine classifiers including; Bernoulli Naive Bayes, Multinomial Naive Bayes, Gaussian Naive Bayes, Linear SVM classifier, decision trees, random forests, extra trees, logistic regression, and adaboost on logistic regression. We found that the random forest and extra tree classifiers performed the best out of the classifiers selected for our dataset.

We tuned various hyperparameters for each of these classifiers, using 10-fold cross validation to detect overfitting and help us determine the best settings for these hyperparameters.

Ultimately, extra trees with 33 trees in the forest performed the best with random forests with 50 trees in the forest coming in second in terms of overall accuracy.

IX. Contributions

Madhav found the data source, then collected and split the data into the training and test set. Madhav also worked on the linear models, including logistic regression. Also played with different ideas of using only verbs and other various modifications. Dhanush worked with the decision trees, random forests, adaboost and the extra trees. Jasmeet worked on the Naive Bayes systems. He was working on the webpage as well; however, we quickly agreed that this was too far from the scope of the project and would require more time than it was worth. Evan worked with the SVM classifier, as well as building early implementations of the webscrapers and API pollers that would gather the most up-to-date information from GDELT, and helped perform further tuning on the extra trees and random forests. Evan wrote this report and curated the data and results available into the plots found herein.

References

[1] C. Healey. (2016) Visualizing twitter sentiment. [Online]. Available: https://www.csc2.ncsu.edu/faculty/healey/tweet_viz/

[2] "The GDELT project". (2016) Events database. [Online]. Available: http://www.gdeltproject.org/

[3] P. Schrodt, *CAMEO Conflict and Mediation Event Observations*, 1st ed. Pennsylvania State University, Department of Political Science, 2012.

[4] R. Schumaker, Y. Zhang, and C.-N. Huang, "Sentiment analysis of financial news articles," 2009.

[5] R. Bolla, "Crime pattern detection using online social media," Master's thesis, Missouri University of Science and Technology, 2014.

[6] B. Liu, "Sentiment analysis and opinion mining," 2012.

[7] D. Haider-Markel, A. Mahalley, and M. Johansen, "Understanding variations in media coverage of u.s. supreme court decisions," *International Journal of Press/Politics*, vol. 11, no. 2, 4 2006.

[8] C. S. Khoo, A. Nourbakhsh, and J. Na, "Sentiment analysis of online news text: A case study of appraisal theory," *Online Information Review*, vol. 36, no. 6, pp. 858–878, 11 2012.