SVM for classifying satirical news headlines

Nolan Tjapkes

ABSTRACT

More than 5000 news articles are published each day. Somewhere in this ocean of information, there are not only articles intended to inform but also articles intended to deceive or entertain. Satirical articles while not intended to deceive sometimes these articles can unintentionally have this effect. We will be applying TF-IDF vectorization and then a SVM classifier to discern between real articles and satirical articles based solely off of the title of the respective article. The data set (Misra and Arora, 2019) was found on Kaggle and consists of a combination of articles sourced from The Onion (satirical) and HuffPost (Real). The dataset has over 28000 datapoints with each data point consisting of the headline, a link to the original article, and a simple satire or not indicator value.

INTRODUCTION

Classifying text based on purpose is something that we frequently do as humans often without thinking about it. Using context, current events, and general social knowledge, it is relatively easy for us to identify sincere vs sarcastic text in most cases. However many of these features that we have access to as humans are not available to a machine model at least not directly. It will be necessary to generate features for the machine models to analyze using the text in the titles since the text alone would lack patterns that common numerical machine models like Support Vector Machines, Artificial Neural Networks, or logistic regression are able to recognize.

This leads us to the field of Natural Language Processing (NLP), we are concerned with the semantics aspect of NLP since we are trying to derive whether or not the meaning of the headline would indicate a sarcastic article or a sincere article. Or at the very least we are unconcerned with the syntax or grammar of the headlines. Once we have used some sort of NLP to translate the headline to some form of data that we can perform some kind of computational analysis on the problem becomes a much simpler classification problem.

A NLP method that seems like it would be a good fit for this data set is the TFIDF vectorization process. This will convert each headline into a vector that gives the relative frequency of a word or phrase that appears anywhere in the data set. While this method won't directly tell us about the meaning of the words or phrases in the headlines, we will be able to analyse the words that appear in both real and satirical headlines and allow the model to identify trends within each class.

PREVIOUS RESEARCH

Text Vectorization

Bengfort Bengfort et al. (2018) describes text vectorization methods from most simple to more modern methods. The simplest model is the bag of words model, which works by taking all of the unique words or tokens in some corpus (larger text) and creating a vector of that length for each individual text to analyse for each time that the corresponding word appears. In the simplest version the value of the vector components for each text is the number of times each token appears. Bengfort mentions that a common improvement made is to divide the count by the length of the text so that the vectors are normalized. Another method is one hot vectorization where tokens found in the text recieve a value of 1 rather than any frequency. This can be beneficial because using the frequency method can elevate the importance of very common words. According to Bengfort this is particularly important for smaller datasets. Finally the TF-IDF method provides both information about the frequency of a word within a text while accounting for the issue of making common letters have too much influence. TFID is the most common method used today.

Sentiment Analysis

Sentiment Analysis is a large and ongoing area of research due to its inherent complexity. Medhat et al. (2014) Medhat surveys the current state of sentiment analysis. She notes that there are 3 levels of sentiment analysis to tackle, at a document, sentence or aspect level. She breaks down the current sentiment analysis methods into 2 main camps, the lexical camp using traditional semantical analysis and statistic and the Machine Learning camp using supervised and unsupervised classifiers. She goes on to discuss the most common methods of feature extraction used and notes that most of them use the bag of words approach discussed above. She also walks through the most common classifiers used to perform sentiment analysis she concludes that SVM and Naive Bayes models are the most common and are used as the baseline that most sentiment analysis model models are compared to.

METHODS

Preprocessing

Most of the preprocessing done to this data will be done in the TF-IDF phase. However a couple small changes were made to the data before vectorization. First, the data was split into testing and training data sets with 80/20 train test split. Given we have 28000 data points and that TF-IDF benefits greatly from a larger training data set this seemed like an appropriate split. Then the train/test labels were changed to

-1 for a negative value (not satirical) and 1 for a positive value (satirical). This will eventually help the accuracy of the SVM model. The first step to prepare text data for almost any NLP is to tokenize the data. Tokenization is the process of splitting the data into tokens which are groups of text. In this case each headline is split from a single string into a list of strings that make up the headline. SKlearn's TF-IDF vectorization command does this automatically. It splits each headline into a list of component words and even removes punctuation from the list. The next step is to remove punctuation and 'stop-words' from the list. Stop-words are words that serve a grammatical role like 'is', 'that' and 'the'. While it would seem that the removal of stop words would eliminate unnecessary features in this case the model was hurt but implementing stop word removal. There are two other methods commonly done for NLP that were experimented with but ultimately decreased model accuracy in this case. The stemming/lemmatization; stemming applies a simple algorithm to each token to reduce it to a 'stem'. The idea is combine terms that carry the same root meaning but different suffixes. Lemmatization is similar; however rather than broadly applying a algorithm; lemmatization is a more complex process that applies vocabulary and morphology knowledge to determine whether or not a word can be conflated with it's stem. For example a stemmer would lead to the conflation of the words universe and university, while a lemmatizer would not. Both of these were manually implemented using NLTK and tested. However, again, neither of these were used since they decreased model accuracy.

TF-IDF Vectorization

Now that we have a list of tokens for each headline, we are able to vectorize each headline. The simpliest vectorization technique is count vectorization: each token anywhere in the data set is assigned a feature and for each list of tokenized words their corresponding feature is the number of times the token appears in the list. This allows us to draw conclusions on which words are used in each context. TF-IDF vectorization works similarly but uses an algorithm to weight words that appear very frequently lower than words that don't. The formula the value of a particular TF-IDF feature is:

$$\frac{TF}{\log(ND/DF)}$$

where TF is the percent of tokens in the token list that are the token correlated to the feature. While ND is the number of lists of tokens (documents) and DF is the number of lists that the token in question appears in. So for each headline we now have a vector of the length of the number of unique tokens in the input data set after prepossessing and for any token in the headline a value to tell us about its relative frequency compared to the total data set, all other values will be 0. SKLearn's module also lets us exclude tokens from all vectors if they don't appear in a certain number of documents and consider combinations of tokens as a single token for vectorization purpose. By using the Grid Search function, the best accuracy was with considering both single tokens and two to three token phrases and excluding those that didn't

appear in at least two documents. Note the TF-IDF vectorization process is 'fit' on the training data set and uses tokens from that data set as the vector features. Also the IDF value is calculated from the set it is fit to alone. So for the test data the features will only consider tokens used by the training fit and the IDF will be the same for each feature as in the training set.

Support Vector Machine

A simple linear SVM model was applied to the TF-IDF vectors. In the SVM (support vector machine) model predictions are made by a decision boundary where one side is mapped to negative values and the other is mapped to positive values, for a linear SVM this decision boundary has the formula

$$C.T * x = 0$$

, where C.T is some vector of coefficients corresponding to the feature vector x. For each x_n vector in the data set it's relative 'prediction' value is equal to -1 if $C.T * x_n < 0$ and 1 if $C.T * x_n \geq 0$. The vectors that satisfy $C.T * x_n = \pm 1$ are the support vectors. Initially the coefficients and bias term can be any non zero value, because values get refined by maximizing the margin between the support vectors.

$$M = \frac{2}{||C||}$$

Here M is the margin between the support vectors which is maximized by minimizing $||C|| = \sqrt{\sum_{j=1}^{n} c_j^2}$. This is the loss function that must be minimized to optimize the predictive power of the SVM model. However there is an issue the coefficients must satisify $y * C.T * x \geq 1$. But for most use cases this isn't possible this is where soft margins come in. For each term we now also include a ξ term to the loss function for when the point is not seperable the ξ term is the distance past the margin that the particular point is. But since we still want to maximize the margin we have some λ that is multiplied by this ξ term so that we can control how 'soft' the final margin should be. The final loss function looks like this

$$||c|| + \lambda * \sum \xi$$

. This loss function is minimized by using gradient descent, where we take the gradient of of the loss function with respect to C and change the values of C by the gradient multiplied by some learning rate that is small enough to ensure that there aren't any jumps past the minimum loss value. In the case of this model, SKlearn allowed for a adaptive learning rate and grid search determined that the optimal lambda was .001.

RESULTS

Overall, the accuracy of the model on the testing data set was around 85 percent. While this isn't perfect, it certainly performs better than the baseline model of guessing which would be 50 percent accurate since the data set is well balanced.

Most of the interest comes from analysing the feature selection and moreover the important features in terms of classification. Below are figures showing the tokens with the strongest correlated coefficients.

Figure 1: Most positive tokens (satire)

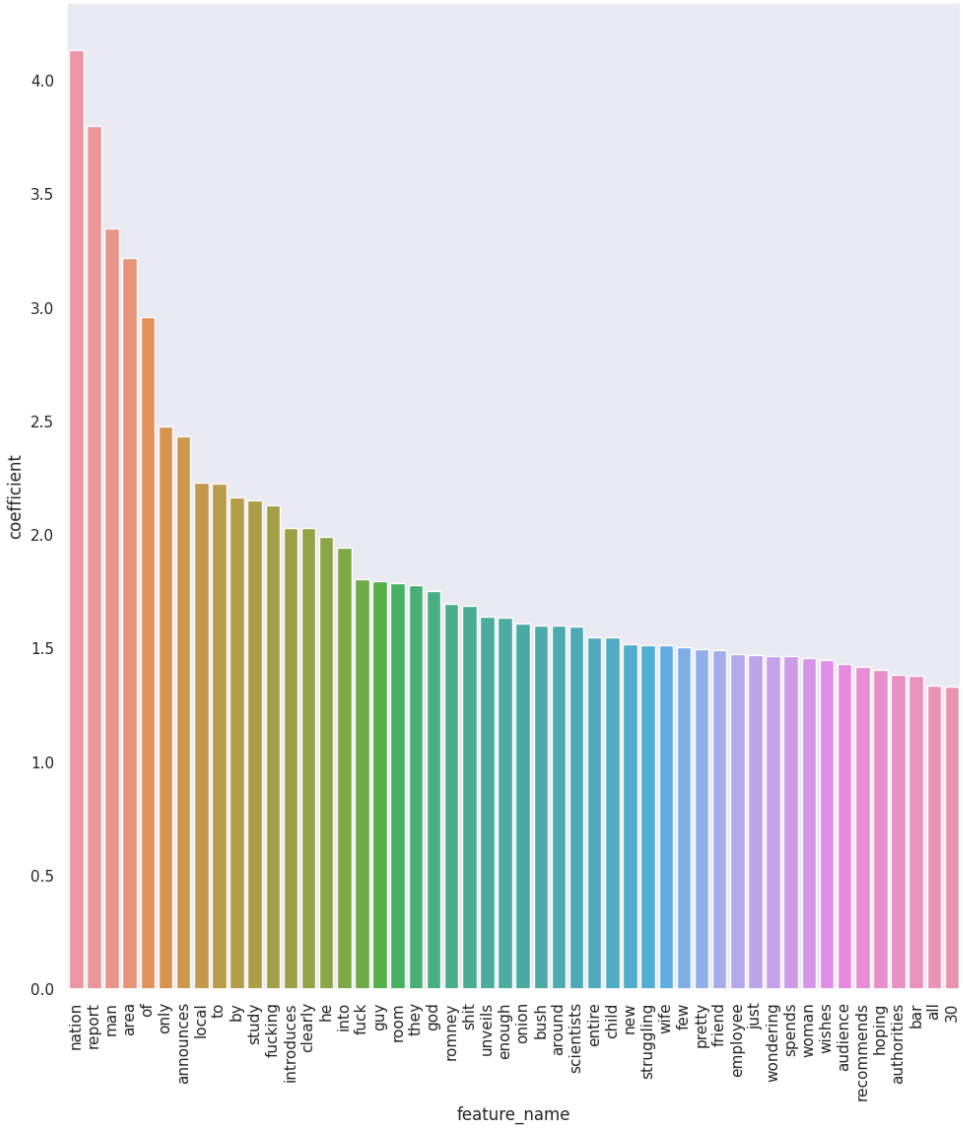
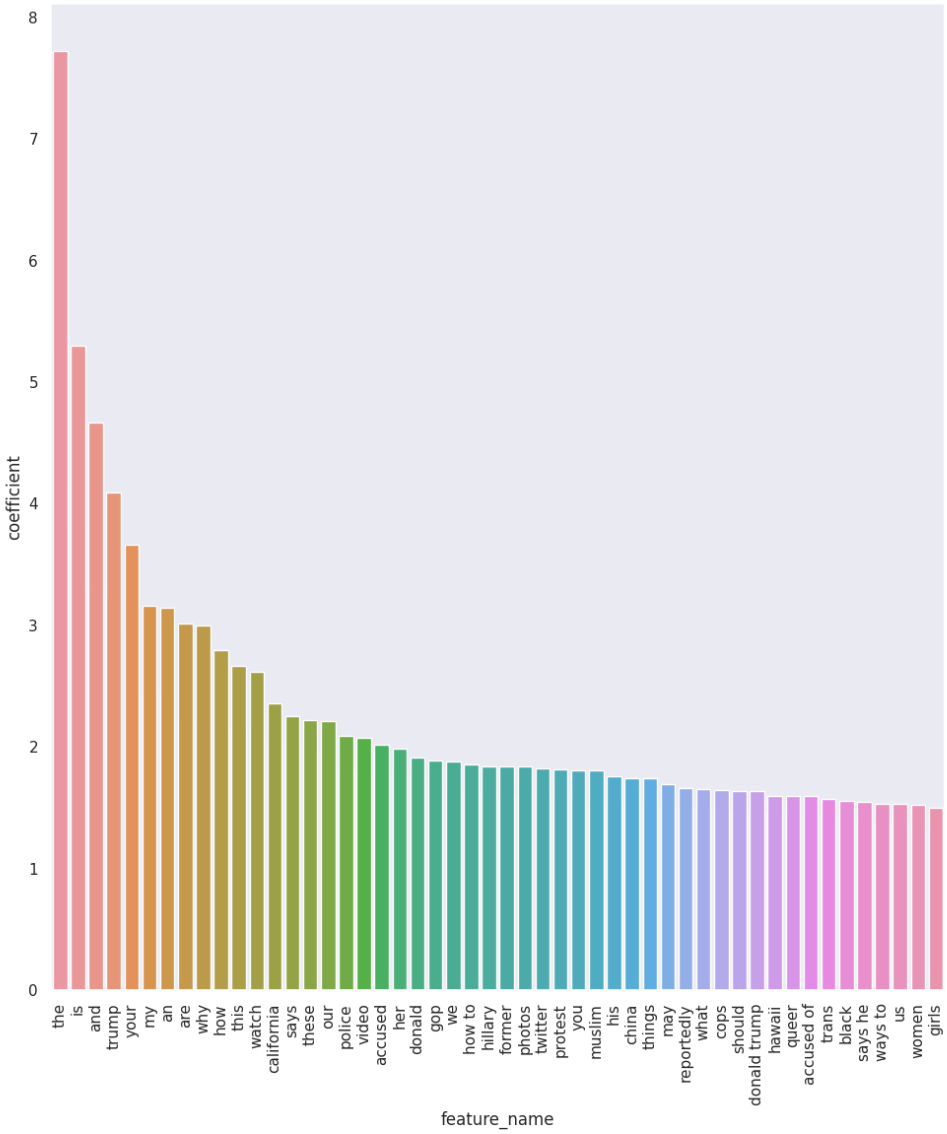
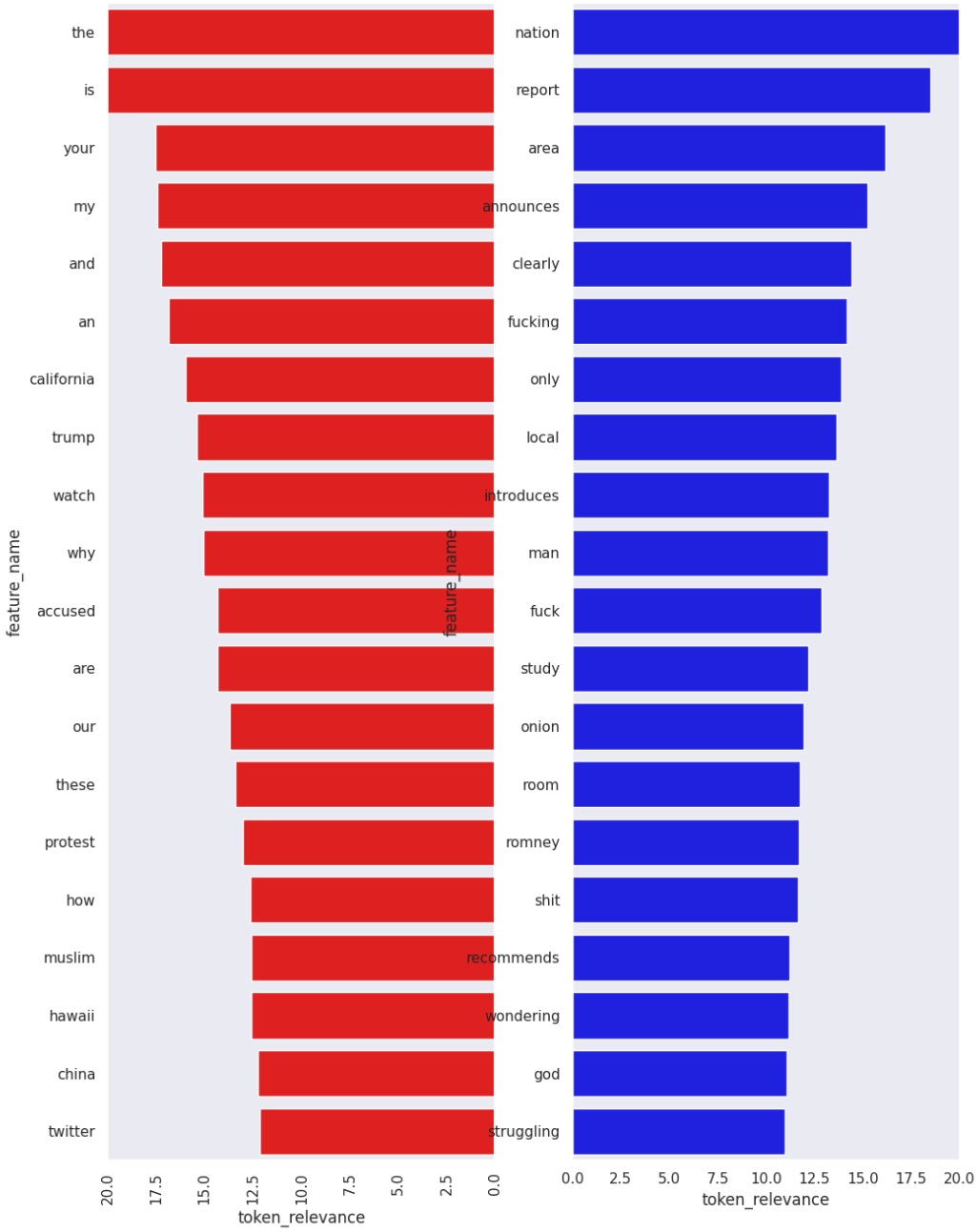


Figure 2: Most negative tokens (authentic)



I had expected the strongest coefficients to be relatively uncommon word/tokens but this was clearly not the case. In fact many of the most strong coefficients were in actually stop words like the, so, etc. In order to try to get some less common terms to appear I tried to sort the features by their coefficient times their IDF (inverse document frequency). The results are contained in the figure below.



It is clear that satirical articles contain more curse words than authentic. It also seems like the sourcing of the headlines(from HuffPost and TheOnion) definitely had

influence on the results. Moreover a lot of the prediction power from this model seems to come from the differences in writing styles between the two sources since many of the most predictive words are stop words.

CONCLUSION

In conclusion, it seems that this model might not be the best for this data set. It seems to make a lot of its predictions based on factors that tell us very little about actual sentiment behind each headline. The accuracy of 85 percent does perform significantly better than a simple baseline but as more non-semantic features of the text are removed the model performs significantly worse.

REFERENCES

Benjamin Bengfort, Rebecca Bilbro, and Tony Ojeda. *Applied text analysis with python: Enabling language-aware data products with machine learning.* " O'Reilly Media, Inc.", 2018.

Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with python.* O'Reilly, 2009.

Gobinda G. Chowdhury. Annual review of information science and technology issn 0066-4200 - strath, 2003. URL `https://strathprints.strath.ac.uk/2611/1/strathprints002611.pdf`.

Walaa Medhat, Ahmed Hassan, and Hoda Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, 5(4):1093–1113, 2014. ISSN 2090-4479. doi: https://doi.org/10.1016/j.asej.2014.04.011. URL `https://www.sciencedirect.com/science/article/pii/S2090447914000550`.

Rishabh Misra and Prahal Arora. Sarcasm detection using hybrid neural network. *arXiv preprint arXiv:1908.07414*, 2019.

Rishabh Misra and Jigyasa Grover. *Sculpting Data for ML: The first act of Machine Learning.* 01 2021. ISBN 9798585463570.

REFERENCES NOT DIRECTLY CITED