Theo Demetriades
November, 2021

ABSTRACT

The following sections describe a recent project in natural language processing, in which a pre-trained transformer neural network (more specifically, BERT) was fine-tuned to determine whether a given body of text was politically left or right-leaning. A dataset of around 90,000 tweets collected using Python was used to train the model, and its validation accuracy in classifying the party of tweets' authors was about 92%. The model's accuracy fell to about 85% when used on (an admittedly small sample size of twenty) online articles. All the data and code is available on GitHub at https://github.com/tdeme/NLP-Political-Bias.

INTRODUCTION

According to the Pew Research Center, "the overall share of Americans who express consistently conservative or consistently liberal opinions has doubled over the past two decades from 10 percent to 21 percent." In other words, the United States is more politically polarized than it has ever been in the last twenty years.

This paper summarizes an independent project conducted for fun over the summer that attempts to analyze partisanship by using natural language processing (NLP) to classify text as politically left-leaning or right-leaning.

The project has five major parts to it:

1. Collect large datasets of tweets judged to be partisan
2. Conduct an initial bag-of-words analysis on these datasets
3. Train a transformer NLP model on the collected data
4. Verify the performance of the model and test on other data
5. Create a tool with the model that can be used by a non-programmer

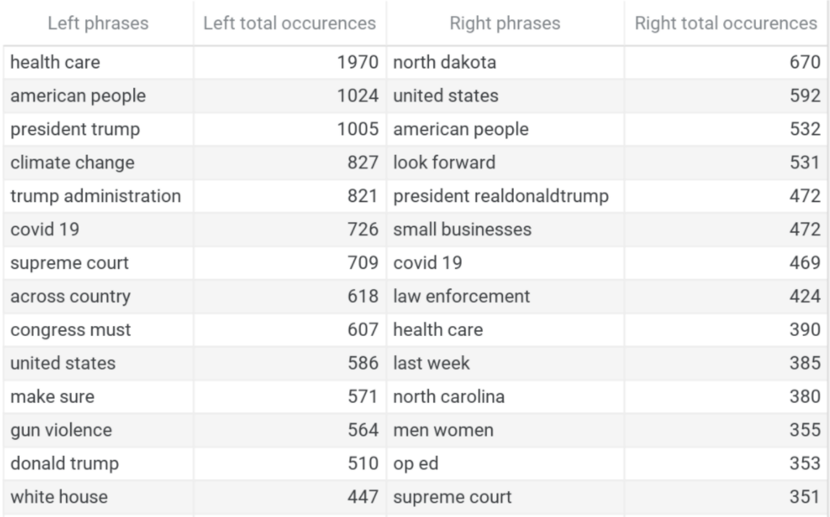
METHODS AND RESULTS

Part 1

The first part of this project was collecting the data. The notebook that was used to collect the data for the project can be found on the project's GitHub page at `Notebooks and Scripts/tweet_scraping.ipynb`. This notebook used the python package twint to scrape tweets from 20 politicians. The govtrack ideology scores were used to determine which politicians should be included for both left and right-leaning datasets. After scraping the tweets with twint, they were saved in the form of csv files and can be found at `Results and Data/left_tweets.csv` and `Results and Data/right_tweets.csv`.

Part 2

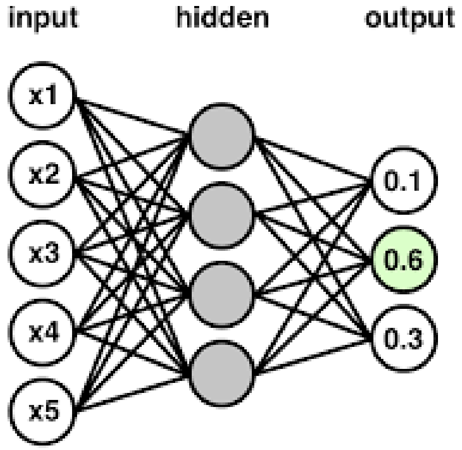
Next, a bag-of-words analysis was conducted on the datasets collected in part 1. The analysis can be found at `Notebooks and Scripts/Bag_of_Words_Analysis.ipynb`, and is inspired by the work done in this article (except using the tweets collected in part 1 instead of speeches). Using scikit-learn's `CountVectorizer`, two more csv datasets were created, which show the most frequent two or three word phrases used in left and right leaning tweets. The following is a sample of those results:



Left phrases	Left total occurences	Right phrases	Right total occurences
health care	1970	north dakota	670
american people	1024	united states	592
president trump	1005	american people	532
climate change	827	look forward	531
trump administration	821	president realdonaldtrump	472
covid 19	726	small businesses	472
supreme court	709	covid 19	469
across country	618	law enforcement	424
congress must	607	health care	390
united states	586	last week	385
make sure	571	north carolina	380
gun violence	564	men women	355
donald trump	510	op ed	353
white house	447	supreme court	351

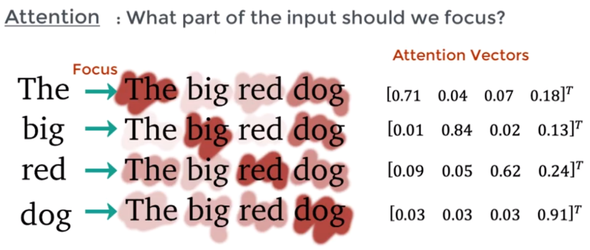
Part 3

At a high level, a neural network can be thought of as a graph in which the nodes represent neurons, and the connections between the nodes represent synapses in a brain. Deep learning algorithms use neural networks to solve problems by adjusting connection strengths between the nodes based on patterns in training data. Deep learning models are said to have multiple "layers," each with its own set of nodes, or neurons. The first layer is the input layer, the last layer is outputs, and all the ones in between are called "hidden layers."



Picture from
https://towardsdatascience.com/coding-up-a-neural-network-classifier-from-scratch-977d235d8a24

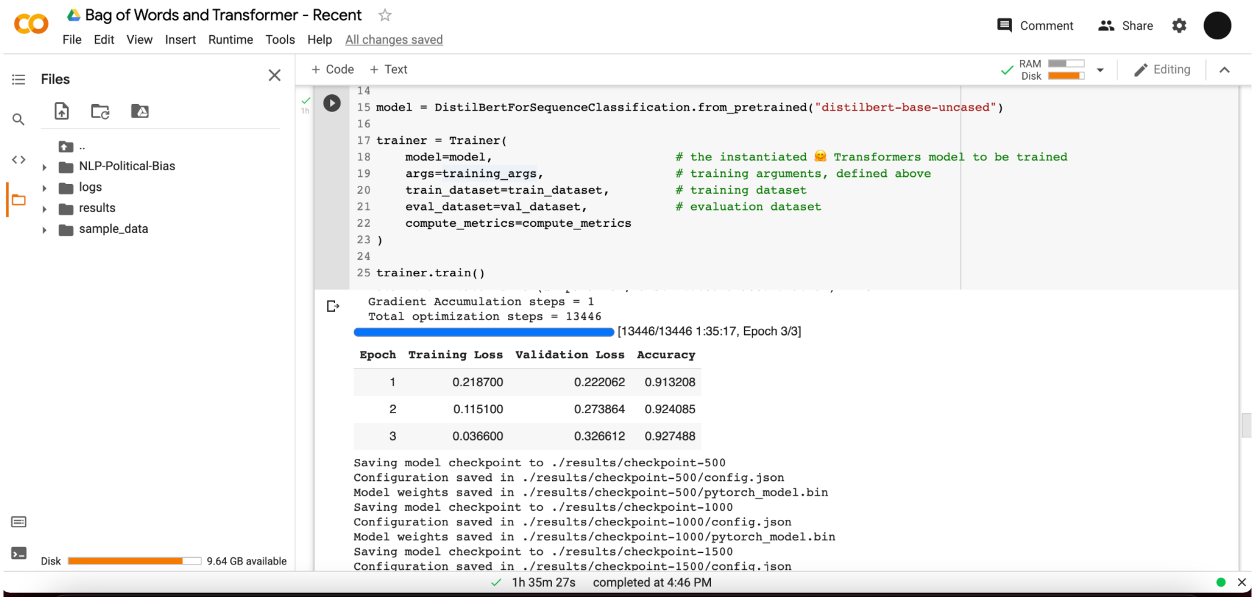
In deep learning models designed for natural language processing, the inputs (words) are turned into "tokens," which are vectors, which are then passed to the model. The type of neural network used in this project is called a Transformer. Using Transformers over the other most common neural network architectures in NLP has two main benefits: they are better equipped to find connections between words that are farther apart, and they can be trained more efficiently (and therefore in less time). The reason Transformers can be trained more quickly is largely because their training is more easily parallelized, which helps take advantage of a GPU processor. Transformers also utilize "attention," by creating "attention vectors" for each token in a sequence, which signifies how strong of a semantic connection there is between that token and the rest of the tokens in the sequence. The following is an example of a visualization of attention in a short sentence. A similar visualization of attention using the model trained for this project can be viewed by running `/Notebooks and Scripts/model_attention_visualizer.ipynb` in a Python notebook.



Picture from
https://towardsdatascience.com/transformer-neural-network-step-by-step-breakdown-of-the-beast-b3e096dc857f

Again this is only a high level overview of how Transformers work. A more detailed description can be found in the 2017 paper *Attention is All You Need*, written by the creators of the Transformer architecture (Vaswani et al.).

BERT, which stands for Bidirectional Encoder Representations from Transformers, is a commonly used pre-trained model in NLP. Pre-trained models have the benefit of being less prone to overfitting, since there is a wider range of training data. Using the collected tweets, and following the Hugging Face documentation for the pretrained Transformer, a BERT model was trained to predict the party of a tweet's author with relatively high accuracy. The datasets can be downloaded by anyone from the GitHub page, and they can train their own model using the code in `/Notebooks and Scripts/model_training.ipynb`, which should result in a validation accuracy of around 92%. The model was trained over 3 epochs, and the other hyperparameters can also be found in the model training notebook. The following is a screenshot of the notebook used to train the model.



```
model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased")

trainer = Trainer(
    model=model,                         # the instantiated 🤗 Transformers model to be trained
    args=training_args,                  # training arguments, defined above
    train_dataset=train_dataset,         # training dataset
    eval_dataset=val_dataset,            # evaluation dataset
    compute_metrics=compute_metrics
)

trainer.train()
```

```
 Gradient Accumulation steps = 1
 Total optimization steps = 13446
```

[13446/13446 1:35:17, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.218700	0.222062	0.913208
2	0.115100	0.273864	0.924085
3	0.036600	0.326612	0.927488

```
Saving model checkpoint to ./results/checkpoint-500
Configuration saved in ./results/checkpoint-500/config.json
Model weights saved in ./results/checkpoint-500/pytorch_model.bin
Saving model checkpoint to ./results/checkpoint-1000
Configuration saved in ./results/checkpoint-1000/config.json
Model weights saved in ./results/checkpoint-1000/pytorch_model.bin
Saving model checkpoint to ./results/checkpoint-1500
Configuration saved in ./results/checkpoint-1500/config.json
```

✓ 1h 35m 27s completed at 4:46 PM

Part 4

The next step was to download the tokenizer and model weights and test them out locally using the `/Notebooks and Scripts/model_testing.py` script. This Python script tests the model trained here using 20 randomly selected articles that were judged to be partisan: 10 left-leaning and 10 right-leaning. To scrape the articles from the urls, the `Article()` object from the Newspaper3k web scraping Python module was used to extract the plain text to be tokenized and finally classified.

Part 5

The end-goal of this project was originally to make a tool that people could use to detect bias in sources of media like articles and tweets. The simplest way to interact with the original PyTorch model is through the general classifier (at `/Notebooks and Scripts/general_classifier.py`). Here, one can either enter the url of an article or plain text, and see the output of the predictions after being put through the softmax function (gets probabilistic outputs based on the logit outputs of the model). While a Chrome extension was developed as well, its performance was unsatisfactory, likely due to an issue with the tokenizer that arose from translating the code to JavaScript.

A website was additionally made for the presentation of this project, which is still currently being developed. Making the site was like a smaller project of its own. First, the Flask Python package was used to make the website locally (Python had to be used instead of something like JavaScript or PHP because the model uses Python), and then a web server was set up and configured locally using Apache.

RESULTS/DISCUSSION

While the model managed to correctly predict the political leaning of the publications of 17 out of 20 articles in `/Notebooks and Scripts/model_testing.py`, its performance is fairly inconsistent. Even though it would have been nice if the model could predict partisanship of bodies of text with an accuracy at or near 100%, there may still be some value in analyzing the results of this project.

No such thing as "bad" results

Instead of simply labelling an experiment a failure if the results are different from what was expected, one can take a closer look at the results to find something of value. In this case, when does the model "work" and when does it "not work"?

Tricking the Model

There are some phrases that seem to go both ways, and it is likely that no significant conclusions can be drawn from these cases. For example, entering “health care” yields a left-leaning prediction, while entering “healthcare” yields a right-leaning one. On the other hand, after some testing, one may notice that the model can be tricked by entering phrases usually used by one party or another. These "partisan phrases" are the same ones found in the table generated during the bag-of-words analysis: For example, entering the phrase "Donald Trump" or "Trump Administration" yields a left-leaning prediction, while the phrase "President Trump" results in a right-leaning prediction. This is probably because when the phrase "Trump Administration" appears in text, it's usually in the context of criticism and therefore more often written by a left-leaning author. On the other hand, the title "President Trump" has a more authoritative connotation, and is used by right-leaning authors more commonly.

Specific Issues

Another insight that can still be gained from entering text directly into the model is which specific issues are more heavily associated with which party. A short list of words and phrases the model deems right-leaning includes abortion, immigration, education, and military, and some ones it finds left-leaning are gun control, LGBTQ/LGBTQ rights, and renewable energy.

SUMMARY

This paper has outlined steps taken to research natural language processing as a tool to examine partisanship in text. First, around 90,000 Tweets from various senators’ accounts were collected to be used as in datasets both to analyze and to use as training data for a BERT classification model. Next, the model was trained up to ~92% validation accuracy and deployed on 20 sample articles, which it classified with an accuracy of 85%. Finally the behavior of the model was inspected by feeding it plain text to classify, and the results were summarized and discussed.

References

Attention is All You Need, by Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, at https://arxiv.org/abs/1706.03762.

Detecting Politically Biased Phrases, article by Jack Bandy, at https://towardsdatascience.com/detecting-politically-biased-phrases-from-u-s-senators-with-natural-language-processing-tutorial-d6273211d331.

Govtrack ideology scores, at https://www.govtrack.us/congress/members/report-cards/2020/senate/ideology.

Hugging Face documentation, at https://huggingface.co/transformers/training.html.

Newspaper3k article scraping package documentation, at https://newspaper.readthedocs.io/en/latest/.

NLTK (Natural Language Toolkit) documentation, at https://www.nltk.org.

Pew Research Center on political polarization, at www.pewresearch.org/politics/2014/06/12/political-polarization-in-the-american-public/.

PyTorch documentation, at https://pytorch.org/docs/stable/index.html.

Scikit-learn Count Vectorizer documentation, at https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html.

Twint tweet-scraping package, at https://github.com/twintproject/twint.