Francis Jo
Nicolas Mackay
Dustin Hayes

Using BERT to Predict Political Affiliation

Abstract:

The purpose of this study is to fine-tune a BERT (Bidirectional Encoder Representations from Transformers) model to predict the political affiliation of a Twitter user. For the purposes of this study, we limited ourselves to two classes: “Democratic Party” and “Republican Party”. Likewise, we limited our study to 885 politicians for ease of access, ease of ground truth labeling, and to limit noise i.e. non-political tweets. We then used the Twitter API to retrieve 1,000 tweets per user included in our study. We then processed this data so as to make it compatible with the BERT algorithm. We attempted a number of different data organizations throughout our attempts. We found the most success in terms of performance and computation time by concatenating tweets by each user into multiple <500 word chunks before processing. We then took the mean prediction among tweets belonging to a user to be the user-level prediction. We ran BERT with a variety of hyperparameter settings, and with different training/validation/test splits. Our highest performing runs were able to correctly identify the political affiliation of politicians in our test set with 85.7% accuracy.

Relevance:

Such a classification system would be useful for exploring the preferences and ideologies of large groups of people interacting on Twitter or other social media sites (Adithya Rao, Nemanja Spasojevic, 2016). Paired with a sentiment analysis model, one could use such a model to track how groups of people with certain ideologies feel about specific topics (Adithya Rao, Nemanja Spasojevic, 2016). Politicians and political organizations could also use a model such as ours to identify users with particular leanings for the purpose of engaging them in political discourse (Stefan Stieglitz, Linh Dang-Xuan, 2012). Such models are also useful in the social sciences as a low-cost and efficient methods of obtaining data relating to the political preferences of a large number of people (Larence Ampofo, Simon Collister, Ben O ‘Loughlin, Andre Chadwick, 2015). Such a model might also be used to identify politicians whose political messaging contradicts their declared party. This information could be useful to voters and/or journalists who wish to characterize a politician by how they present on social media as opposed to their self-declared status.

Data Collection:

The starting point of our data collection process was a Kaggle dataset containing 2515 Twitter user IDs, the names of the politicians associated with them, the political affiliation of those politicians, and a number of other features such as age and birthplace that we considered superfluous for our purposes. Before utilizing this dataset to collect the information that we needed from the Twitter API, we cleaned it in a variety of ways. First, we removed rows with missing values in the fields we were interested in. Then we removed rows corresponding to users that identified as neither Republican or Democrat, as we limited our scope to just these two classes. We did take any party affiliation with that included the words “Democrat” or “Republican”. For instance “The Democratic Party of California” became “Democratic Party” in our analysis.

A large volume of tweets per user will generate a data set that will be pre-processed for BERT modeling and classification. This large volume of tweets came from the Twitter API (Application Programming Interface). The API provides access to programmatic data from Twitter. This data contains in depth information regarding tweets, users, and profiles. Many programming languages can utilize API for data mining. For the study, python code was used to gather and process data from Twitter.

To have access to this data, a user must create what is called a Developer account (Dev account). Signing up is just like signing up for a Twitter account, but with more verification. This account comes with 3 crucial tokens that are used in python to connect to the API. The Bearer token, Access Key and Secret Access Key. The Bearer token is what authenticates requests from the Dev account. This allows for basic access and requests such as gathering tweets and profiles with all associated data. The Access Key and Secret Access Key act as a username and password for the Dev account. This will give you more freedom when data mining but requires an authentication command. Limits are associated with a dev account. A user at the most basic level of access can only request 500,000 tweets every month. Collection for this project utilized two different accounts and over 800,000 tweets were accumulated.

Mining these tweets only required a bearer token. This allowed for python commands such as ‘search_recent_tweets’ and ‘get_user_tweets’, as well as many others. Since the model examines a large number of tweets for each user, the primary command that we used was ‘get_user_tweets’ ,which accepts a User ID and outputs the up to 100 most recent tweets, unless the dates in which you are looking for tweets is specified. We used this function both to find tweets by user ID, and to filter our initial Kaggle dataset so as to exclude inactive accounts. ‘get_user_tweets’ accepts an optional argument named ‘start_time’. ‘start_time’ specifies the earliest date that we wish for our tweets to have come from. By setting the start date to October 1st, 2022, we were able to filter out user IDs that had not posted since then, as ‘get_user_tweets’ returned nothing in this case. After all stages of filtering had been completed on our initial Kaggle dataset, we were left with 885 user IDs. It was quite common that a politician had one or more accounts they had long since abandoned.

The simple form of the ‘get_user_tweets’ command only returns up to 100 tweets at a time. Incorporating a paginator in the command made it possible for us to reach our goal of 1,000 tweets per user. Next, we were ready to input the list of remaining user ids from the Kaggle data to create a new data set with 1,000 tweets for each politician. The finalized data contained information for Tweet_ID, User_ID, Text, and Political Party. See figures for how data was retrieved and output from python.

```python
def get_paged_tweets(user_id):
    """Script for gathering raw tweets for a list of user id and saving them to csv"""
    df = pd.DataFrame(columns=['TweetID', 'UserID', 'Text'])
    for tweet in tweepy.Paginator(client.get_users_tweets, user_id, exclude='retweets',
                                  max_results=100).flatten(limit=1000):
        # remove tab
        text = tweet.text.replace('\n',' ')
        #append raw tweet
        df.loc[len(df.index)] = [tweet.id, user_id, text]
    return df
```

Figure 1: How tweets were collected for each user ID. This function was called with all 885 user IDs.

	TweetID	UserID	Text
0	1597430067260698624	816181091673448448	https://t.co/ev2jRb5Gwr
1	1596911731682332677	816181091673448448	📢 Every American deserves high quality health c...
2	1596519144609837057	816181091673448448	Happy #SmallBusinessSaturday! What small busin...
3	1596156765984342018	816181091673448448	Ahead of the winter months, the Biden-Harris A...
4	1595779274304200704	816181091673448448	Happy Thanksgiving! I hope you enjoy a relaxin...
...	...	...	...
995	1416144069890351105	816181091673448448	It was a pleasure to host a dedication service...
996	1416098058929119240	816181091673448448	#ChildTaxCredit payments began on July 15! I w...
997	1415806002226679810	816181091673448448	Happy Birthday to Maggie Lena Walker! A Richmo...
998	1415414564657762305	816181091673448448	Read more on this important project. ⬇️ https:...
999	1415414562422198280	816181091673448448	I am proud to have secured funding for @VCU's ...

Figure 2: Example of 1,000 tweet output.

Data Processing:

We initially organized our Twitter data with a collection of 885 csv files, each corresponding to a particular politician. Each row of these files contained the text and tweet ID of a particular tweet, as well as the corresponding user ID for the politician in question. Included below is a sample csv corresponding to former president Barack Obama.

Figure 3: 1,000 tweets by Barack Obama organized in a csv.

Such an organization was prudent during the initial stages of our project. It was at this point that we realized that Twitter had failed to receive all 1,000 tweets from 135 politicians. Only the data for those users where all 1,000 tweets had been recovered were considered, effectively reducing our data set to 1,000 tweets by 745 politicians. We wrote a simple script to take each tweet from those users where all 1,000 tweets were gathered, along with tweet ID, user ID and party affiliation, and organized the resulting data in one large csv file. Each tweet corresponded to one row of this csv file. We had removed new line characters by this stage of the project, but most of the text-cleaning was performed by the BERT tokenizer at a later stage.

We attempted a variety of different organizational schemes as we gained experience working with our BERT model. Running BERT with all 745,000 tweets proved very time consuming. We needed to take only a subset of our data. Additionally, BERT can only handle units of 512 tokens. Longer texts would be truncated such that only the first 512 tokens were considered. Lastly, we needed to consider the balance of the dataset. After cleaning the data and failing to pull all 1,000 tweets from some of our users, we were left with only 316 Republican users out of 750 users in total. We created adequately small data sets in the following ways:

- Take a balanced subset of tweets evenly from each politician. For example, we might only take 100 tweets from 316 Democrats and 316 Republicans with the intention of including more tweets after successful runs.
- Chunk tweets into groups of approximately 10 to 15 tweets and run each chunk through the BERT model as one unit. Consider 5 to 13 chunks per user. Balance the dataset such that the same number of chunks were taken from Democrats as Republicans. This

option tended to be faster than the previous option, even with the same number of tweets overall.
- Chunk tweets into groups of approximately 5 to 10 per user instead of 10 to 15 per user. This option was used as a middle ground between the two previous options.

Tweets were selected to be included in tweet chunks based on length. We operated on the assumption that longer tweets might be more analytical in nature, and therefore more likely to contain some political argument or politically minded statement. In an attempt to reduce noise, we gave priority to the longest tweets first when constructing our tweet chunks. The large tweet-chunk option was nearly always faster than the other two options, and tended to produce better results. The authors of this paper are still unsure as to why chunking tweets seemed to produce better results, or whether or not those better results were due to some combination of luck and hyperparameter settings as opposed to any real effect.

Data Analysis Method: BERT Model

BERT (Bidirectional Encoder Representations from Transformers) is one of the recent machine learning techniques for natural language processing (NLP). It was first released in November 2018 by researchers at Google AI Language (Muller, 2022). The model was pre-trained using the sparsely large text data based on BookCorpus with approximately 800 million words and Wikipedia with approximately 2.5 billion words (Winastwan, 2021). Before BERT was released, other NLP models could only read texts or sequences of words in only one direction, either from right to left or left to right. On the other hand, BERT reads text in both directions to fine-tune embeddings for context, which makes it a powerful tool for NLP (Winastwan, 2021). It uses layers called “Transformer encoders” and “attention heads” to extract features from text data, hence the name of the model. Some applications of BERT include sentiment analysis, where texts are classified into categories by predictions, and question answering, where relevant answers to our questions are found given some texts (Muller, 2022). There are two distinct BERT models: Base and Large. The BERT Base model consists of 12 layers of Transformer encoders, 12 attention heads, 768 hidden size, and 110 million parameters. The BERT Large model consists of 24 layers of Transformer encoders, 16 attention heads, 1024 hidden size, and 340 million parameters (Winastwan, 2021). Each model contains a lot of parameters to estimate, so it can take a long time to train a BERT model, especially with a large dataset.

The BERT model requires that each input be tokenized in a specific format. Each input must begin with a token called the “classification token”, which is denoted as [CLS] (Nishanth N., 2020). Each input in the system must have an equal number of tokens, so padding tokens, denoted as [PAD], are added to each tokenizer if its length is too short (Muller, 2022). As padding tokens are not part of the original input text, BERT will mask those tokens and will not attempt to fine tune their embeddings. Finally, a token called [SEP] is added to indicate the end of each sentence (Nishanth N., 2020). The tokenized inputs can then be used by the BERT model.

When processing, each input goes through multiple layers of both Transformer encoders and attention heads to extract features. In the BERT Base model case, the tokens go through the 12 layers of Transformer encoder x 12 attention heads = 144 combinations to extract features (Vig, 2019). These features are obtained and converted into 768 dimension embedding vectors that are extracted from the [CLS] tokens and used to classify categories at the end. The diagram by Ruben Winastwan (Figure 4) summarizes the procedure (Winastwan, 2021).

The extracted features can be used as inputs for any appropriate classifying technique, such as logistic regression or linear neural networks. In logistic regression, inputs are directly used to predict or classify binary outputs. In linear neural networks, inputs enter hidden multiple layers that perform linear transformations of the inputs and use these transformed inputs to predict or classify outputs (DeepAI). We found the most success using a linear neural network for classification.

To optimize results for particular use cases, a process called fine-tuning is used on the BERT model. Fine-tuning is the process of finding the parameters of a model that need to be adjusted so as to perform best with a particular dataset. The BERT model was already pre-trained with the large text datasets and it has its own parameters by default, but attempting to use the pre-trained embeddings for classification could result in poor performance. The full dataset is first split into training, validation, and testing datasets. The training and validation data sets are then used to adjust the internal parameters of the BERT model to better fit our use case. Finally, the model is evaluated by prediction accuracy on the testing data.

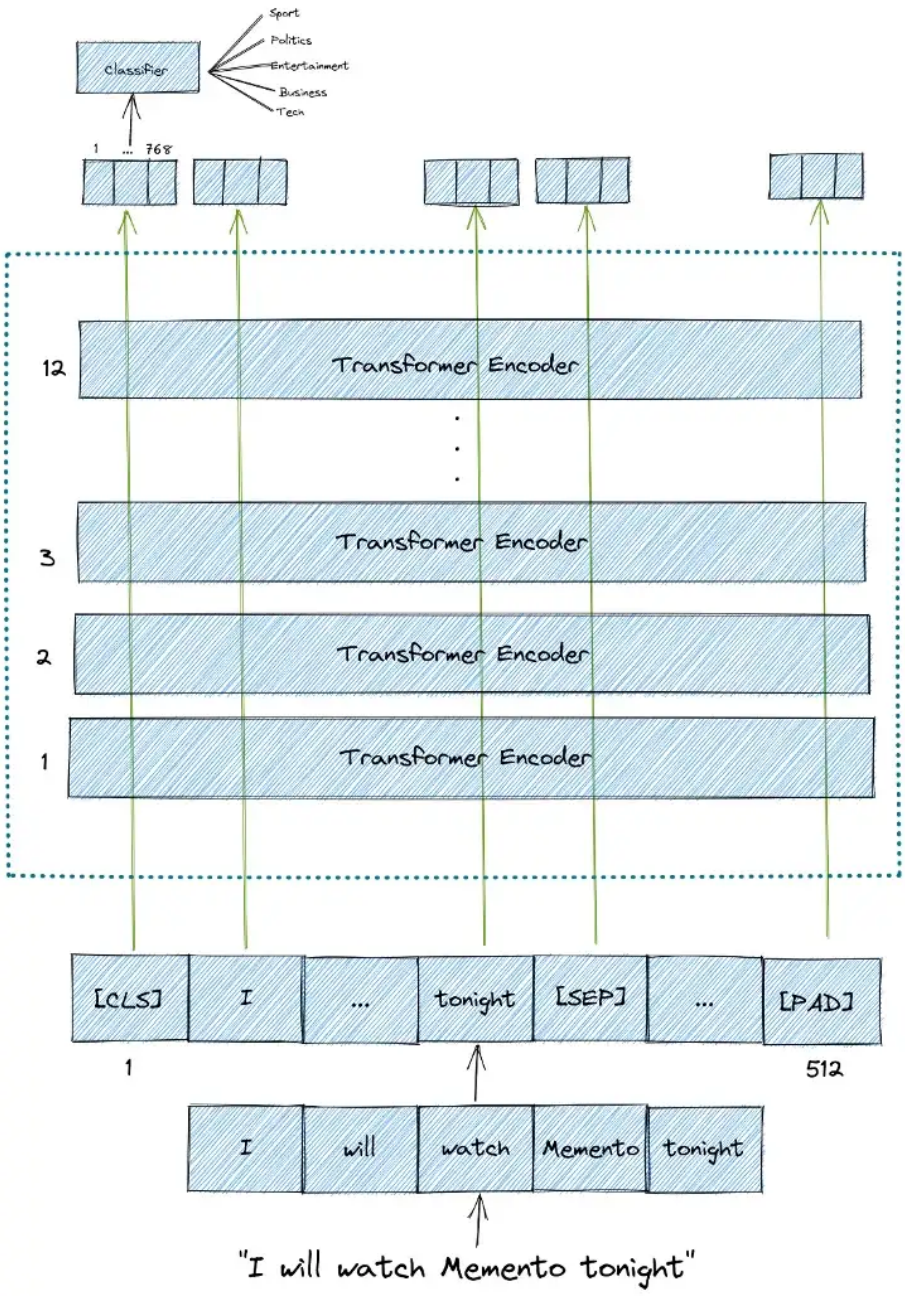

Figure 4: Ruben Winastwn's visual representation in BERT

Results:

A number of hyperparameters, including learning rate, epoch and batch size, greatly affected the quality and reliability of our results. Included here are the results of three of our more

successful runs, along with the problems associated with each that motivated us to search for better implementations of the BERT model. The coding in each output is as follows: **{0: Democrat, 1: Republican}**.

Run 1:

- Learning rate: 1e-6, Epochs: 5, Batch Size = 2, Split (training/validation/test): 80/10/10
- Data split method: Chunk by tweets such that no unit contains more than 500 tokens. 5 chunks per user.
- Data profile: 632 politicians. 316 Republican and 316 Democrat.
- Chunk-wise Test Accuracy: .837; User-wise Test Accuracy: .857, Chunk-wise Training Accuracy: .916
- Hardware: 2017 Macbook Pro.
- Run time: ~8 hours.

		Prediction	
Actual		0	1
	0	25	1
	1	8	29
	Score accuracy		0.85714
	HM		0.86360
	Ave Class accuracy		0.87266

Figure 5: Confusion Matrix for Run 1. Calculated at user-level.

User-wise accuracy was calculated as follows: take some arbitrary user X. Assume that X's tweets were organized into 5 chunks. Also assume that 3 of these chunks were classified as belonging to a Republican. The user-level prediction for X is then Republican. User-wise accuracy is then the number of correctly predicted users divided by the total number of users in the test set.

Comments on Run 1: Due to our lack of parallel computing functionality, we were consistently forced to run with less data we would have ideally. This problem was a consistent feature among our runs. As pointed out by Dr. Haiyan Wang of Kansas State University, our split was not very appropriate for this type of problem. We used a 50/20/30 split on later runs.

Run 2:

- Learning rate: 5e-5, Epochs: 5, Batch size = 128, Split (training/validation/test): 50/20/30
- Data split method: Chunk by tweets such that no unit contains more than 500 tokens. 6 chunks per user.
- Data profile: 632 politicians. 316 Republican and 316 Democrat.
- Chunk-wise accuracy: .754; User-wise accuracy: N/A
- Hardware: One core of the Beocat at Kansas State University.
- Run time: ~10 hours.

Comments on Run 2: Just as with Run 1, we were not able to train with as much data as we would have liked. Other problems resulted from our inexperience with Beocat. We accidentally overwrote some of the output, including that which we would need to calculate user-level accuracy. Even if we had saved this output, we errantly created 6 chunks for each user. An odd number of chunks would be required to calculate user-level accuracy in the manner previously described.

Run 3:

- Learning rate: 1e-6, Epochs: 5, Batch Size = 2, Split (training/validation/test): 50/20/30
- Data split method: Chunk by tweets such that no unit contains more than 500 tokens. 7 chunks per user.
- Data profile: 632 politicians. 316 Republican and 316 Democrat.
- Chunk-wise Test Accuracy: .800; User-wise Test Accuracy: .857, Chunk-wise Training Accuracy: .910.
- Hardware: 2017 Macbook Pro.
- Run time: ~10 hours.

		Prediction	
Actual		0	1
	0	86	13
	1	14	77
Score accuracy		0.85789	
HM		0.85727	
Ave Class accuracy		0.85742	

Figure 5: Confusion Matrix for Run 3. Calculated at user-level.

Comments on Run 3: Run 3 was intended to be a recreation of Run 1, the only differences being a more appropriate split and additional data. Epochs, batch size and learning rate were all identical to Run 1. There is still not as much data as we would have liked. This run displayed some level of repeatability, as prediction accuracy for the test set was almost identical to Run 1; Run 3 had a user-wise test prediction success rate of .8578, whereas Run 1 had a user-wise test prediction success rate of .8571. Interestingly, we observed more balance in terms of incorrect labeling with Run 3 than we did with Run 1. In the case of Run 1, nearly all of our mislabeled users were Republicans labeled as Democrats. We observed no such effect in Run 3.

Conclusion:

Despite a number of problems that we were unable to solve in the time allotted for this study, we were able to generate enough signal to increase prediction accuracy over a baseline level of 50% by approximately twenty to thirty points on our most successful runs. Run 1, despite its inherent problems, produced some interesting results. For this run we produced a user level prediction vs. label csv, where each user in the test set is identifiable by user ID. A subset of our output is shown below in figure 6. Run 2, which was performed on the Beocat, was moderately successful and helped us determine the relationship between batch size, learning rate, and the overall performance of the model. Run 3 represents our ability to recreate the success of Run 1 with a more appropriate split and more data.

```
,UserID,Label,Prediction
0,1080870981877534720,1,1
1,816303263586914304,1,1
2,1082380458976051202,0,0
3,1052896620797460481,0,0
4,20744708,0,0
5,234053893,1,0
6,996094929733652481,1,1
7,876894308309147650,1,0
8,817076257770835968,0,0
9,815241612154417152,1,1
10,403581958,1,1
11,827279765287559171,1,1
12,914815520842616834,1,1
13,1078771848882593793,0,0
14,806583915012046854,1,1
15,1067818539179024386,1,1
16,2461810448,0,0
17,28946201,1,1
18,76132891,1,1
19,76452765,1,1
20,15824288,0,0
21,242426145,0,0
22,25375137,1,1
23,828977216595849216,1,1
24,15764644,0,0
25,160010755,1,0
26,242883110,0,0
```

Figure 7: Labels and Predictions for a subset of the Test set, Run 1.

As part of the analysis of our results, we went back and manually inspected the incorrectly classified users and their tweets. Although we will not consider each incorrectly classified user here, let us consider the user corresponding to the first mislabeled class, with a user ID of 160010755.

User 160010755 is the 84 year old former advisor to president Nixon, John Dean. He was predicted as a Democrat in Run 1. In the Kaggle dataset which we used to generate our ground truth labels, he was reported as a Republican. Unfortunately, this mistake was propagated to our data. Mr. Dean is not a Republican; he is a former Republican who currently identifies as Independent (Barabak, Mark ,2017). In fact, he often speaks out against Republican policy, and is/was especially critical of George W. Bush and Donald Trump. A few of his more telling tweets are included below:

Example Tweet 1; Tweet ID: 1579604410581540864; Author: John Dean

- “Donald Trump could not and cannot pass a basic civil service examination to get a job with the federal government!”

Example Tweet 2; Tweet ID: 1572361518335012864; Author: John Dean

- “Trump and GOP enablers are an evil that must be addressed for what it is doing to our nation. The way to deal with it is stay informed for only fools want what Trump is offering.”

Example Tweet 3; Tweet ID: 1569536373048356865; Author: John Dean

- "Remarkably, Donald abused his power as president at every turn! So so much worse than any President ever!!”

It should not be surprising that our classifier failed to identify John Dean as a Republican because he is not a Republican; more importantly, he does not, in the estimation of this author, post on Twitter as if he was. We observed, unsurprisingly, that the model had difficulty predicting politicians who posted less clearly partisan content. Less surprising still, and comfortingly on some level, the model struggled with politicians that, unbeknownst to the authors of this paper, were incorrectly labeled from the start. Out of 745 politicians, 2 have been identified that were mislabeled. In the future, a more rigorous examination of the quality of our starting data is required.

There were multiple issues which we were not able to adequately solve in the time frame available to us. Perhaps most critically, we did not include parallel computing functionality or GPU compatibility in our code, and as such were unable to fine-tune our BERT model at a reasonable speed with as much data as we would have liked. Continued study necessitates additional computing power so that we can use much more data to train and evaluate the model. Quicker run-times would also allow for us to find better hyper-parameters, as repeated runs were difficult and time consuming.

One question that still remains unanswered is how well our methods would generalize to non-politician users. We focused on politicians as the data was readily available to us, their political affiliation is known and well documented, and it is safe to assume that many of them will frequently post about politics. Users from the general population presumably will, in general, post about politics much less, and as such will produce much noisier data. Were we to develop this model to work for any user, not just politicians, we would need to expand our training set to include general-population users. We might also consider developing a second tier of classification to filter out tweets by whether or not they contain political information before turning them over to our BERT model for party classification.

Lastly, we would like to mention that we more frequently observed Republicans misclassified as Democrats than the other way around. Run 3 might be considered an exception in this regard, as the balance of incorrect classification was nearly even. The dataset was balanced such that we always had an equal number of tweets by Democrats and Republicans. We are unsure of what caused this effect, or if this effect was observed by chance. More runs and investigation is required on this front.

In summary, we were able to use BERT to predict the political affiliation of politicians on Twitter with accuracy approximately 20 to 35 points better than a random guess on our most successful

runs. Specifically, we observed two runs with a user-wise accuracy of .857 and one with a chunk-wise accuracy of .754. A number of issues created difficulties for us, most notably our lack of the computing power required to train BERT with more data in a reasonably short period of time. A lack of quality and up-to-date information in the dataset we used to label our politicians was also an issue. Further study is required to remedy these issues.

Citations:

Rao, A,; Spasojevic, N. (2016, July 13). Actionable and political text classification using word embeddings and LSTM. *arXiv.org*.
https://arxiv.org/abs/1607.02501

Stieglitz, S. (2012). Social Media and Political Communication: A social media analytics ...
https://www.researchgate.net/publication/306158429_Social_media_and_political_communication_a_social_media_analytics_framework

Ampofo, Lawrence; Collister, Simon; O'Loughlin, Ben; Chadwick, Andrew (2015): Text mining and social media: when quantitative meets qualitative, and software meets people. https://hdl.handle.net/2134/27593

Samoshyn, A. (2020, November 23). US politicians Twitter dataset. Kaggle.
https://www.kaggle.com/datasets/mrmorj/us-politicians-twitter-dataset

Muller, B. (2022). BERT 101 State Of The Art NLP Model Explained. *Hugging Face*.
https://huggingface.co/blog/bert-101

Winastwan, R. (2021). Text Classification with BERT in PyTorch. *Towards Data Science*.
https://towardsdatascience.com/text-classification-with-bert-in-pytorch-887965e5820f

Nishanth N. (2020). Question Answering System with BERT. *Analytics Vidhya*.
https://medium.com/analytics-vidhya/question-answering-system-with-bert-ebe1130f8def#:~:text=Question%20Answering%20System%20using%20BERT&text=For%20the%20Question%20Answering%20System,embeddings%20and%20the%20segment%20embeddings.

What is a Hidden Layer?. *DeepAI*.
https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning#:~:text=In%20neural%20networks%2C%20a%20hidden,inputs%20entered%20into%20the%20network.

Fine-tune a pretrained model. *Hugging Face*. https://huggingface.co/docs/transformers/training

Dreiseitl S., Ohno-Machado L. (2002). Logistic regression and artificial neural network classification models: a methodology review. *Elsevier*.
https://www.sciencedirect.com/science/article/pii/S1532046403000340#:~:text=Compared%20to%20logistic%20regression%2C%20neural,pruning%20the%20network%20after%20training.

Barabak, Mark (2017). *On politics: John Dean helped bring down Richard Nixon.* Now he thinks Donald Trump is even worse. *Los Angeles Times*. https://www.latimes.com/politics/la-pol-ca-on-politics-column-20170602-story.html

Vig, Jesse. (2019). Deconstructing BERT, Part 2: Visualizing the Inner Workings of Attention. *Towards Data Science*.
https://towardsdatascience.com/deconstructing-bert-part-2-visualizing-the-inner-workings-of-attention-60a16d86b5c1