Twitter Sentiment Analysis: Comparison of Old School vs. New School Methods for Natural Language Processing

Michael Wise[*] Advisor: Eitel Lauría[†] Marist College - Honors Thesis

December 16th, 2022

Abstract

This project focuses on the topic of finding the best technique for performing sentiment analysis on Twitter data, specifically related to the Ukraine vs. Russia Crisis. We explore the practicality of using social media, like Twitter, as a mode of identifying sentiment of the general population during a specific time period. In traditional NLP, sentiment analysis comes with many problems, including procedures for collecting/scraping data, preprocessing it, and the dilemma of dealing with unlabeled text. In combination with a lexicon called VADER, we aim to show an effective methodology for deriving a sentiment score and assigning feelings and emotions to tweets and messages using both "old-school" and "new-school" techniques, with the latter referring to state-of-the-art transformers. While showing the differences between old-school techniques (stemming, stop words) and new-school techniques (transformer tokenization & self-attention) for preprocessing, we describe a paradigm for what we think is the most effective procedure for processing social media data. Through various validation techniques, we address the most up to date and innovative transformer encoders like BERT, RoBERTa, ALBERT, and ELECTRA to determine what is the best at processing natural language and predicting sentiment from that data in comparison to and old-school multinomial Naive Bayes classifier trained using a bag of words approach. For our purposes, RoBERTa performed the best with 89% accuracy on unseen tweets, while ELECTRA showcased its ability to fine-tune the quickest without losing too much accuracy at 87%. Transformers are shown to perform better than old-school methods with little to no preprocessing or hyperparameter tuning required, as the multinomial NB was only able to achieve 80% accuracy.

[*]School of Mathematics and Computer Science, Marist College, Michael.Wise1@marist.edu
[†]School of Mathematics and Computer Science, Marist College, Eitel.Lauria@marist.edu

Contents

1 Introduction

Language is the basis of nearly all communication within our society. It is one of humanity's greatest inventions. In fact, there are over 7,000 distinct languages spoken worldwide, each with their own style and structure. Most of these languages have many irregularities and ambiguities in their grammar, which just makes human communication even more impressive. It raises a question of whether other beings, or perhaps, machines, would be able to interpret text and extract some meaning from it. This is exactly what the goal of natural language processing is.

1.1 Natural Language Processing

There are many sub-fields of natural language processing (NLP), but they all have the same basic idea. The ultimate aim of NLP is being able to read, understand, and decode text, most commonly in the form of human words. Hours of research and implementation go into developing computer models with the goal of having them understand text and language as well as humans do. One example of an NLP task is speech recognition, with the primary goal of translating voice data into text data. The most difficult part of speech-to-text is being able to decipher different accents, pronunciations, and dialects of various languages. Everybody talks with different emphasis and intonation, making it tough to make sense of what is being said. Another area of NLP is sentiment analysis. In contrast, the goal of sentiment analysis is to derive certain qualities, aka "sentiment", from a string of text. It is focused on extracting things like attitudes and emotions.

1.2 NLP in Our Daily Lives

Whether you believe it or not, every technological aspect of your life in some shape or form utilizes a component of NLP. Look at your phone - Siri uses a combination of speech recognition and NLP to process any request you might have. These "voice assistants" like Siri, Alexa, and Cortana all can process exactly what you say in a matter of seconds. Have you ever wondered how these technologies work? We are able to communicate with chatbots that eerily resemble human conversation, along with the ability to program them to detect actual emotions and sentiment.

1.3 The Twitter Problem

On the topic of communication and sentiment, it is no secret that social media has taken over as arguably the most prominent form of communication, even over face-to-face interaction for some. Social media is very powerful - it allows us to express our feelings and attitudes on certain topics instantaneously. One of the most popular social media platforms today is Twitter. There are hundreds of millions of Twitter users active every day, with a diverse representation of people from all over the world. Because of its large user base, many consider Twitter as a decent measure of public opinion (while skewed). If we could find a way to extract data from Twitter, filter it by certain keywords that pertain to a topic, and then perform sentiment analysis on it, we could get a good measure of how people feel. We could also derive a way to classify tweets by sentiment. By assigning labels to tweets, we can then use various machine learning methods to predict sentiment on any set of tweets.

1.4 Addressing the Elon Musk Controversy

When initially beginning this research, there was no news of change in Twitter ownership or policies, but since then, Mr. Elon Musk completed a $40+ billion dollar buyout of the social media platform, laying off large numbers of employees. The company is nearly unrecognizable from what it started as back when this project began in the Spring of 2022. At the current moment, the company is experiencing massive negative cash flows, and many previously restricted or banned users and features have since been reinstated in attempt to promote free speech.

This thesis is not about the politics or drama behind the new ownership of Twitter or its status staying afloat as a company, but rather to demonstrate the potential of NLP using social media as a case study. For now, the same technologies for scraping and querying data work on Twitter, but please account for potential changes in the platform in the coming months. Whether Twitter, Facebook, Instagram, or whatever social media platform is used doesn't really matter, as the main point is to showcase the contrast between “old-school” and “new-school” procedures for processing and analyzing natural text data. More specifically, emotions and feelings about certain topics are more polarizing than ever before on the platform, so sentiment analysis has its place in being able to truly depict the emotional state of what The New York Times describes as a “global town square”. [1]

2 What is Sentiment Analysis?

The primary task of sentiment analysis is to determine whether textual data is positive, negative, or neutral. What we actually define in terms of a scale for sentiment is endless, and as we will see, it is very subjective depending on how you define it. Yet, we will stick with the three main categories as mentioned before, as it is the practice found in most literature. We briefly will touch on some other types however.

There are quite a number of different types of sentiment analysis that can be done. Like mentioned before, we can focus on the polarity of the text (positive, negative, neutral). However, many models go beyond this, and are able to detect specific feelings and emotions, like whether someone is happy, sad, or angry. Other use cases include imperativeness (urgent, not urgent), and even intent. The use cases for sentiment analysis will only increase exponentially as continue to build more and more accurate models for processing this data.

We will concentrate on polarity for this investigation, but leave the door open for other potential characteristics to extract in the future.

2.1 Considerations for Irregularities in Sentiment Analysis

Consider the following line of text: “Netflix has the best selection of film”. It is quite easy for humans to tell that this is a positive message, and most basic methodologies for determining sentiment score wouldn’t have any trouble deducing that either. However, now if we look at a sentence like “I do not *dislike* horror movies”, we now have an example of negation (in this case a double negative). These types of phrases don't come as naturally to a machine processing all of the words. Human language is filled with all of these kinks that sway the meaning of phrases and sentences. This is a common challenge we have to consider. Luckily, a lot of the lexicons that can derive sentiment score are trained for these weird cases that would otherwise be difficult to categorize.

2.2 Tweets as a Mode of Sentiment

For this study on Twitter data, we need to look at a couple of things. First, we want to decide on what specific data we want to obtain. We can filter out key words related to a certain topic. For sake of relevancy, we chose to look at tweets pertaining to the conflict between Russia and Ukraine. Great, we have a topic, but how do we get any of this data? The second consideration is the actual method of scraping the Twitter data. The easiest and most reliable method for doing so is by using Twitter's API. [2] There are a couple caveats with Twitter's API - it is quite limiting. There is a strict amount of tweets you can extract (about 500,000 per month on the free version for developers). Of course, opportunities to extract a lot more data are available, but require payment, and frankly, for the analysis in this paper, we won't be needing millions of records anyways due to the nature of these newer algorithms.

Note that there exists some open-source scrapers that have been developed with a lot more freedom. They usually bypass the constraints of the Twitter API. The only issue is that they are constantly getting patched and are ultimately untrustworthy. However, there is one Python package that was found to be very effective, and has no interaction with the Twitter back-end. This is the `snscrape` package, which is actually compatible with many of social media platforms. [3] The implementation relies solely on the advanced queries that can be searched on any browser. It is only limited to how fast the algorithm can load each page of Tweets, along with extracting their metadata. For example, if we wanted to query Tweets related to the Death of Queen Elizabeth between September 8th and 12th, we could do the following in Python:

```python
import pandas as pd
import snscrape.modules.twitter as sntwitter
import itertools
```

```
from tqdm.auto import tqdm

query = "queen elizabeth -filter:replies until:2022-9-12 since:2022-09-08"
tweets = []
limit = 5000

for tweet in tqdm(sntwitter.TwitterSearchScraper(query).get_items(), total=limit):
  if len(tweets) == limit:
    break
  else:
    tweets.append([tweet.date, tweet.username, tweet.content, tweet.id])
queen_tweets_df = pd.DataFrame(tweets, columns=['date', 'username', 'text', 'id'])
```

The `query` string object is the same string that is produced when you use advanced search on Twitter, so the extent of your search is the same as if you were browsing actual Twitter.

For our analysis, we originally wanted to try a bunch of different polarizing topics to analyze, but quickly realized that the predictive classification models all performed very similar, regardless of the topic of the tweet. Thus, we will stick with text focused on the Ukraine vs. Russia crisis for this paper. There also exist data sets on Kaggle that are designed purposefully for sentiment analysis. They are updated on a consistent basis and don't require a scraper, because the authors did all the work for you. Some data sets are actually classified by sentiment from the getgo (we will use some later to validate our model for sentiment score). Regardless of whether someone went in manually to determine polarity, or if the data set was generated artificially, labeled data is really easy to work with if your goal is to make future predictions. If you have pre-labeled data, it is quite easy to train a classification model. In our case, there is no labeling, and no easy way to tell the sentiment of every tweet. In the next section, we will talk about the methodologies used for dealing with this unlabeled data.

3 Natural Language Processing with Unlabeled Data

3.1 Data Preprocessing: Old School Methodology

All tweets were exported to a `.csv` file with various features. The keywords that were used to find these tweets were the following: "ukraine, russia, putin, nato, war". For this example, around 20,000 rows of tweets were picked up. A lot of these columns that were extracted aren't particularly interesting for our purposes, but we will talk about the more interesting ones.

The `tweet` column contains the actual text of the tweet that a user wrote. This is the most important column. We also have some useful information, like the `likes_count`, `retweets_count`, and `hashtags`, which are all pretty self explanatory. In future avenues, it would be interesting to see if we could integrate the number of likes and retweets into our models to get an even more intricate sentiment classifier. The other columns serve more as metadata, like the `username`, `date`, `time`, and `language`. The language is actually important, as we configured the API to pick up more than just English. Unfortunately, unless we were to do translations, the lexicon used only supports English, so we filtered out any records in other languages. There were, however, a good representation of languages - and of course, language corresponds heavily with what country the tweet originates from, which also could play into polarity. Especially when dealing with Ukraine vs. Russia data, people's opinions and sentiments could vary drastically from country to country. Anyways, leaving just English, we are left with 17,307 tweets to work with. Null values were also checked, and it was found that there were 0 missing values. Our data is in a form where we can now safely manipulate some of the text.

3.2 Stop Words & Stemming

Utilizing old school processes, we require a lot of preprocessing of text just so it is at a state to be read by simple models. This calls for removing any "hiccups" that would remove from our analysis when we ultimately tokenize each word in our text. Later on, we will see that this is not that necessary when working with transformers. Using the `NTLK` Python package in addition to `re`, we are able to clean out a lot of our tweets. We can eliminate things like stop words, typos, emojis, and punctuation, as they tend to give us a lot of trouble. For the sake of making things more simple, we can also do things like making every word lowercase, removing usernames, and any funky blocks of text that are left over from encoding. Note that some emojis and punctuation can actually contribute to the overall sentiment, but the models were using for the time being don't really support this. Perhaps we will consider leaving certain punctuation/emoticons in later analysis.

One thing we can utilize to improve the accuracy of predictions from models is something called stop words. These are any word a stop list which are filtered out (i.e. stopped) before or after processing of natural language data. There is no single universal list of stop words used by all natural language processing tools. We can really choose whatever we want our stop words to be. We will just use the base stop list that `NTLK` offers. Some examples of stop words include "its", "an", "the", "for", and "that". They are often prepositions or articles, that while are important for communicating verbally as humans, don't really carry much significance in many NLP tasks, including sentiment analysis. These aren't the words that are going to relate to the meaning or emotion of a tweet or phrase.

Another technique is stemming. It is quite useful when dealing with words that have suffixes and/or extraneous text that often refer to nothing more than context or tense. By removing the suffixes, words are reduced to just their root, which will subsequently reduce the size of our word list and make our models more accurate. In other words, we are reducing a word to its base word or stem in such a way that the words of similar kind lie under a common stem. The stemmer chosen was the Snowball Stemmer, which is seen as a better and updated version of the infamous Porter Stemmer. [4] All of these come from the default dictionaries within the `NLTK` Python library. Below is some examples of what words turn into once they are stemmed.

One issue is that once we stem, sentences become a lot harder to read for humans. A lot of the integrity of English grammar is lost, but it is beneficial and shown to be much easier for computers to understand.

Original Word	Stemmed Word
cared	care
university	univers
fairly	fair
easily	easili
singing	sing
sings	sing
sung	sung
singer	singer
sportingly	sport

Table 1: Result of stemming certain words via the Snowball Stemmer

3.3 VADER & TextBlob

There exists many models that work for prelabeled sentiment analysis, but we opted to choose ones that have historically shown to be accurate at deriving sentiment score to a high degree of accuracy. The two models that were chosen for sentiment analysis were VADER (for Valence Aware Dictionary for sEntiment Reasoning) and TextBlob. VADER was coined by C.J. Hutto and Eric Gilbert, and is a rule-based lexicon that does surprisingly well at deriving sentiment scores for social media data. [5] TextBlob was independently created by Steven Loria, a senior software engineer from New York. The algorithm from TextBlob is a lot simpler than VADER, but it still is quite fast and usable to this day for simpler tasks. [6]

For determining polarity of a string of text, VADER computes a compound score. The score is computed by summing the valence scores of each word in the lexicon, adjusted according to the rules (more explicitly described in the paper), and then normalized to be between -1 (most extreme negative) and +1 (most extreme positive). The compound score is the most useful metric if you want a single unidimensional measure of sentiment for a given sentence or tweet. Thus, this what we will be using as our main measure of polarity. Because it is normalized and the most weighted, it is the most trustworthy.

In a similar fashion, TextBlob will output sentiment on a similar polarity scale, from -1 to +1. This begs the question though, how similar are these scales, and which performs better? To do this, we will have to compare these models to some sort of "gold standard" dataset that already has prelabeled sentiment classes.

4 Choosing a Model for Labelling Text Data

In this section, we will be taking VADER and TextBlob, applying it to an existing data set of tweets that are prelabeled, and comparing what they predict to the actual sentiment labels. To start this process, we require said data set. There exists many public data sets out there that contain labeled Twitter data. We will try two datasets, one being Sentiment140, which is already Twitter data, along with the IMDB data set of movie reviews.

4.1 The "Gold Standard"

In order to evaluate the accuracy of models and make sure our tweets our labelled accurately, we have to compare these sentiment models to some kind of benchmark. A "gold standard", if you will - some corpus of data that we know for certain is associated with either a positive or negative sentiment. Many of these data sets exist specifically for sentiment analysis, it is more of a matter of choosing some that fit the context of the data we are trying to classify.

4.2 Sentiment140 Data Set

The Sentiment140 data set contains over 1,600,000 tweets extracted using the Twitter API. A subset of the tweets were annotated, and then using distant supervision, labels were assigned based off of "positive" or "negative" sentiment (keep in mind that they did not assign any tweets as "neutral"). [7]

Distant supervision is a machine learning technique for collecting training data. Traditionally, one can label data manually by using human annotators. This works well only for small sets of data. Getting humans to label targets is costly in both time and money, as well as the fact that humans are full of error. Human choice and bias is stochastic at times, thus a lot of training data using this approach ends up noisy. As a different approach, one can make use of existing label data, which may or may not be a small subset that was originally annotated by humans. We then generate more training data based on this original data, extrapolating based off of relationships between certain words and their labels. [8] Distant supervision then artificially creates data, usually with a bit of noise/variance thrown in. In general, the authors labeled tweets with positive emoticons such as ":)" as positive, and tweets with negative emoticons as negative. This is a very simplified explanation, yet also addresses the obvious problem of there just being better ways to label data nowadays. This is how the Sentiment140 data set was created. More information regarding the generation of this data is described in the paper [7], as it is not the main focus of this report. It still should be important to note that there might be flaws in the labelling of Sentiment140, as it was created in 2009, before a lot of newer NLP techniques were found.

4.3 IMDB Data Set

The IMDB data set is a collection of movie reviews from IMDB that was compiled by Stanford's AI Lab, designed to be a "benchmark" or "gold standard" for sentiment analysis. They provide a set of 25,000 highly polar movie reviews for training, and 25,000 for testing. Totaling in at 50,000 rows, the IMDB data set gives us a really good standard to use in terms of very polarizing negative and positive data. The only thing that is different from Sentiment140 is the difference in rows. [9] We won't use all 1,000,000 rows of Sentiment140, so this is no big deal. Another thing to note is that one data set is move reviews and the other is tweets, in which the latter will produce a more reliable metric for our analysis.

4.4 Running VADER & TextBlob on Benchmark(s)

Before we try VADER and TextBlob on this data set, we have to consider that the threshold for what is considered positive and negative may vary between the labelling techniques from the data set and the rules set by VADER's lexicon. It is also important to remember that this data set does not include a neutral label. Thus, we initially have made VADER and TextBlob's cutoff to be the following:

- Positive sentiment: compound score ≥ 0.5
- Negative sentiment: compound score ≤ -0.5

Now this is making some pretty massive assumptions. While both models output a polarity score from [-1,1], how do we know how these scores are distributed? We can test this by graphing the distribution of sentiment scores if we run the models on 50,000 rows of each data set. This gives us insight both on the scales of the models and the true polarity of the text data that was picked out in the context of what the models predict.

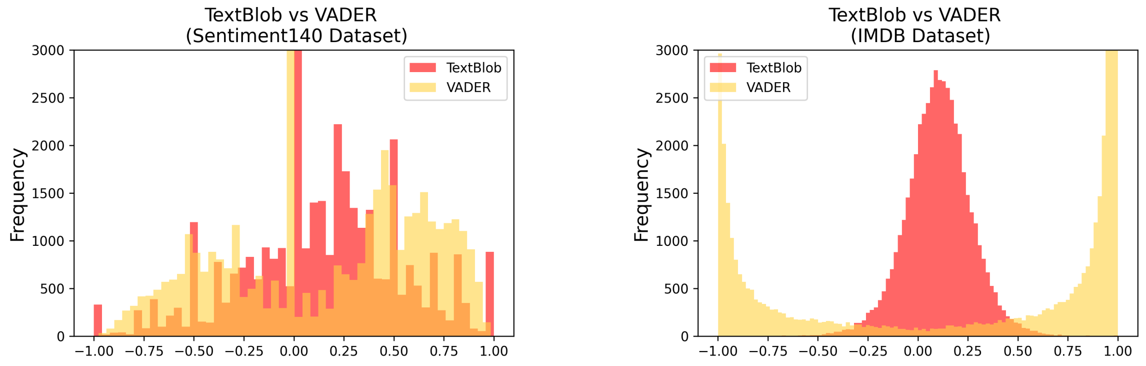

Figure 1: Distribution of Sentiment Scores Comparison between VADER and TextBlob for both benchmark datasets

This shows us a couple of things, both about our models and our benchmark data. Sentiment140 seems to cover a broad spectrum of polarities, some of which are very subjective. This begs the question of whether its original labels are as right as they say they are. And while Sentiment140 and IMDB were supposed to focus on very polarizing messages only, it's clear to see IMDB definitely has a more fair/expected distribution of positive and negatives based off of the sentiment labelers' predictions. What is different however is how scores are assigned.

VADER predictions favor towards the absolute negative (-1) and absolute positive (1), which we should expect if these are supposed to be the absolute worst movie reviews compared to the absolute best. In contrast, TextBlob follows a Normal distribution with a quite small variance, not really outputting scores past .50 in both the negative and positive direction.

We could try a bunch of techniques like scaling the data to try to get these scores to be distributed similarly on the same scale. In fact, if you do standard scaling, remove neutral values (i.e. values where either predicted neutral based on the [-0.5, 0.5] threshold) and compare the percentage of times where VADER and TextBlob agree with each other, you find that for IMDB they agree 88.8% of the time, and 90.1% of the time for Sentiment140.

Another component we care about is speed of model prediction. Fun fact, we tried to use another model called Flair, which utilizes RNNs & LSTM. While accuracy was technically a few points better, it was incredibly slow at deriving sentiment score, and when we have thousands of tweets we need to process in seconds, we found it not worthwhile in comparison to the speed of both VADER and TextBlob. Interestingly, we can look at how fast it takes those two models to come up with a sentiment score for prelabeling our data:

	Sentiment140 (small tweets)	**IMDB (large reviews)**
VADER	9 sec	2 min, 7 sec
TextBlob	15 sec	1 min, 7 sec

Table 2: VADER vs. TextBlob Speed to derive sentiment score on 50,000 rows of each benchmark set

Note that each Tweet in Sentiment140 is a lot smaller than the massive reviews in IMDB, so we expect IMDB to take longer. It is interesting how VADER is a bit faster when working with smaller strings of text, but TextBlob is significantly faster when dealing with larger strings.

In the end, we decided that VADER is the better choice. We can sacrifice a little speed for our context, because we are just using tweets which do have a character limit. In the future, it would be interesting to try an ensemble of labelers to get better opinions, but for now, just using VADER is more than fine for our purposes.

We will omit anything that VADER considers “neutral”, to turn it into a binary problem. Tweets and reviews that are recorded in the middle as neutral can often go either way, so it is best to remove them to not add any bias or noise. This removes about 500,000 rows from our data frame. This keeps things relatively simple. The Sentiment140 data frame uses “0” and “4” instead of “negative” and “positive” respectively. This relationship seems injective (one-to-one), as even though they list everything on a 0-4 scale, there only exists two binary classes being “0” and “4” within the data set. Accordingly, we will rename the targets so it is easy to compare later on.

The next part is actually using VADER to derive sentiment. By instantiating VADER’s `SentimentIntensityAnalyzer()` in Python, we can derive the compound sentiment score. Finally, we take all of the predictions (VADER) and compare them to the true values (according to Sentiment140). This metric is accuracy, denoted by

$$\text{Accuracy} = \frac{\text{\# of Correct Predictions}}{\text{\# of Total Predictions}} = \frac{TP+TN}{TP+TN+FP+FN}$$

where TP and TN mean true positives and true negatives, while FP and FN mean false positives and false negatives respectively. Calculating the accuracy score, we get that VADER is about **80% accurate** in predicting the correct label on Sentiment140 and **86% accurate** on IMDB. Considering that we have over 1,000,000 records for Sentiment140, 80% is pretty decent, and is promising that we can use VADER for unlabelled data.

In fact, one fallacy noticed with Sentiment140 is that when reading some of the tweets verbatim, some of the assigned sentiments do not really match the actual tweet (at least in our opinion, this is also what the tweets look like after they are preprocessed and certain things are taken out of them). If we had a decent sized data set of fully human-annotated data, we think VADER would be even more efficient at determining the same sentiment, given some of the tests run in its original paper. This also could account for change in accuracy - luckily, the IMDB data set derives its preassigned polarity from the actual rating the user left in complement with their review. Therefore, it’s definitely going to be more accurate.

Since we stick with VADER, we should briefly touch on the compound sentiment score it derives. Compound sentiment score is useful for researchers or any data scientists who would like to set standardized thresholds for classifying sentences as either positive, neutral, or negative. The standard, typical threshold values are:

- Positive sentiment: compound score ≥ 0.05
- Neutral sentiment: (compound score > -0.05) and (compound score < 0.05)
- Negative sentiment: compound score ≤ -0.05

For now, we will use this threshold for our classification of our Ukraine data. The `pos`, `neu`, and `neg` scores are ratios for proportions of text that fall in each category. These are the most useful metrics if you want to analyze the context & presentation of how sentiment is conveyed or embedded in rhetoric for a given sentence. We personally didn't look into this feature too much, but it definitely has some use cases.

4.5 Running VADER on Ukraine Data

With the same instance of `SentimentIntensityAnalyzer()` from the previous section, we can produce polarity scores for each of the 17,307 tweets from the valence-based lexicon. Given the threshold from Section 3.3, the compound scores can be assigned to class labels. The distribution of the tweets resulted as follows:

Sentiment (based on VADER)	Number of Tweets
Negative	8695
Positive	4767
Neutral	3845

Table 3: Results from running VADER's Sentiment Intensity Analyzer on the Ukraine data

From the table above, we can see that 8,695 out of the total 17,307, or about 50% of all the tweets were detected as negative. The number of negative tweets nearly doubles the amount of positive tweets. If we remove the neutral class entirely, then 65% of the tweets were identified as negative, meaning that in general, people are overwhelmingly upset in this data set. This ideally makes sense, as we would expect most of the tweets relating to the Ukraine vs. Russia conflict to be negative due to most people being upset with the situation. The "overall sentiment" of the tweets is negative, which gives me much faith in humanity, as we don't picture many people empathizing with Russia. Now that we have data to work with, the next stage is training a model through machine learning to see if it could classify more of these types of tweets, with no context other than the tweets themselves. We will look into the idea of classification models for sentiment analysis. Models like these are what helps us predict and understand where we truly are in the world.

Moving forward, we will describe one classifier that was trained on our data, as well as discuss some state-of-the-art algorithms - these little models called transformers. The ultimate goal is to build a model that outputs the highest accuracy, giving us the highest change of correctly predicting sentiment of tweets. We will describe the advantages and disadvantages of both old school and new school approaches.

5 "Old School" Sentiment Classification

5.1 Bag of Words & Naive Bayes

When you think of even the most basic machine learning algorithms, you quickly realize that most of them cannot take in a raw string of text, but rather some sort of numerical data. Through NLP, we have developed various ways to extract features from these sentences and make them meaningful to a ML model. The method described here is called a bag of words [10]. It is one of the most popular models, as well as one of the most simple models due to its reliance on rudimental Bayesian probability.

Since we have to extract *something* numerically from the text, what if we accounted the occurrence of each word in each tweet? Thus, the whole idea is that the frequency of occurrence words from a "dictionary" that we create ends up mattering in helping classify sentiment. This dictionary (which we can denote as V) simply contains every word that can possibly occur in any of the tweets. The number of documents, or tweets in this case (denoted as N), ultimately serve as the rows of a matrix containing the number of occurrences of a word for each of those tweets. This matrix is of size $(N \times |V|)$, and is called a Document Term Matrix (DTM). The DTM is how we can visualize and implement the bag of words.

	Word_1	Word_2	Word_3	$\ldots$	$\text{Word}_{\lvert V\rvert}$
Tweet_1	0	2	1	$\ldots$	0
Tweet_2	1	0	0	$\ldots$	3
$\vdots$	0	0	2	$\ddots$	0
Tweet_N	0	1	1	$\ldots$	0

Table 4: Example DTM for a bag of words implementation with dictionary V and N number of tweets/observations. Each number lists the frequency of that specific word.

Table 4 shows an example of what the bag of words might look like. A thing that is important with this approach is that we do not care what the *order* of the words is - that is irrelevant to us. We only are concerned about the frequency. Using a `CountVectorizer()` within sci-kit learn, we can separate the text into individual words (what we call tokenization). The most basic tokenizers find spaces between characters, and know to split them into vectors of particular words.

Now with the bag of words in hand, we need a way to classify each of our tweets based off of these frequencies. The Naive Bayes (NB) classifier is a simple classifier that relies on Bayesian probability and the naive assumption that feature probabilities are independent of one another given each class. More specifically, we can use a multinomial distribution (generalization of the binomial distribution in probability & statistics). Consider a document d_i, which is a single tweet in our case. This tweet d_i belongs to a class c_j (eg. "positive", "neutral", or "negative" sentiment) and is made up of a bag of words extracted randomly from a multinomial distribution of event probabilities $(p_1, p_2, \ldots, p_{|V|})$. These events represent occurrence N_{it} in tweet d_i of a word w_t (t is just the index of where a word appears in the dictionary V). We will not go much deeper into much of the mathematics, but the premise is that we then provide conditional categories ("positive" or "negative" sentiment) for each document to fall into based off of whoever has the highest probability. From this model, we can build a simple, yet effective Multinomial NB classifier.

In our implementation, we just run the `CountVectorizer()` in plain vanilla. We could change some of the tokenization around if we want, but keep in mind that our text data was already preprocessed through the use of filtering, stemming and stop words. Similar to what we did when comparing VADER to the Sentiment140 dataset, we are going to drop all of the ”neutral” values, turning this into a binary classification problem. Again, neutral is one of those sentiments that don’t tell us all that much, and we would much rather ignore them for the time being. Eliminating the neutral tweets brings us down to about 13,400 tweets. Remember that we have a much higher distribution of negative tweets (nearly 2:1 ratio). Hence, there is a little bit of class inbalance in favor of “negative”, but it shouldn’t hamper our model accuracy too much. If one sentiment class was significantly less common than another, then we might consider resampling methods. Again, we don’t worry about this too much.

5.2 Training & Evaluation of Multinomial NB with BOW

For evaluating model performance, we will use train-test split. We decided to split the data into 75% training data and 25% testing data. We then fit and transform the training data to our `CountVectorizer()`, putting it in the bag of words form we need so the model can understand the text based off of frequency. Using the `MultinomialNB()` classifier in sci-kit learn, we fit the training data to the model. We then used the trained model to make predictions based off of the 25% remaining test data.

Because this is a binary classification problem, we can derive a handful of predictive performance metrics that evaluate our predictions. Below is a breakdown of the 4 metrics.

Metric	**Score**
Accuracy	80.45%
Precision	80.99%
Recall	91.12%
False Positive Rate	8.88%

Overall, the bag of words in combination gives us 80% accuracy when it comes to predicting sentiment. This is really solid, and a lot higher than initially anticipated from an approach like this. There exists a lot of other classifiers in addition to ways to extract the features from these tweets that have shown to be even more accurate.

5.3 Other Classifiers for Sentiment Analysis

The approach used above utilized vectorizers. We created a single feature vector using all the words in the vocabulary of the tweets. There is an alternate approach called word embedding. Rather than counting frequency of words in each document, we are now capturing aspects of each word, along with their relationships and similarities to other words. Word embeddings open the door for other classifiers, one big one being neural networks. We could use 1-Dimensional convolutional neural networks to classify text (1-Dimensional because of the size of our feature matrices). Another type of neural network that has shown success is the use of recurrent neural networks (RNNs). These types of models consider sequential data, which completely changes the playing field, as tweets are sentences that have words that do follow a certain order. While with the bag of words we ignored the order, we actually can account for the sequence of words in a tweet. One type of RNN is called LSTM (Long short-term memory) which uses feedback connections that can make predictions based on time series (i.e. embedded) data. LSTMs and RNNs, however, are slowly becoming less and less relevant as even better algorithms are being created. We're not going to try any of these, as we are more focused on the latest techniques with transformers, but it's important to acknowledge their existence.

Arguably the most exciting model for NLP tasks has to be transformer-based machine learning techniques. In specific, Google has recently published a model in 2018 called BERT (Bidirectional Encoder Representations from Transformers) [11]. Like recurrent neural networks, transformers can also handle sequential input data, which makes it a great choice for translation, text summarization, and sentiment analysis. The transformer has a variable number of encoder layers and also uses a concept called self-attention. We will briefly explain how these work, and then fine-tune them on our Ukraine data.

6 Overview of Transformers

Let's go back a few years to 2017, where researchers at Google published a paper, proposing an architecture for modeling sequential text data that outperformed recurrence and convolutions in accuracy and cost of training. This model follows the structure of a novel neural network, and is called the **transformer**. [12]

6.1 Introduction to Transformers

In order to understand transformers at a basic level, it is important that we describe the encoder-decoder framework. This concept is not new, and is something that you can see utilized in recurrent neural networks, which used to be the state-of-the-art. Specifically, these models were designed for the task of machine translation, where the goal is mapping a sequence of words in one language over to another. This uses the now fundamental *encoder-decoder* (sequence-to-sequence) architecture, in which the encoder takes the input sequence and converts it numerically into a vector called the *hidden state*. Some information is fed back into the model itself in a feedback loop, until we get to the *final hidden state*. That of which is then passed into the decoder, which is able to generate the desired output, which can be anything - not necessarily just a translated piece of text.

As a way to help the decoder extract context from each hidden state of the encoder, we use a mechanism called *attention*. With attention, we are able to assign a certain amount of weight to each state of the encoder to pick out which input tokens are most relevant for our desired task. According to Tunstall et al. "there still was a major shortcoming with using recurrent models for the encoder and decoder: the computations are inherently sequential and cannot be parallelized across the input sequence". [13]

This is where the recurrent model started to reveal its flaws, and where transformers take on a new paradigm: *self-attention*. With self-attention, we can allow attention to operate on all of the states in the same layer of the neural network. Both the encoder and decoder are given their own self-attention mechanisms. At the end of that process, the output is fed to a feed-forward neural network (FFNN) to then do whatever task we need. The final issue is, how do we train this model if we don't have access to a large corpora of labeled text data to train on? This is where *transfer learning* comes in.

6.2 History of Transfer Learning

Transfer learning actually started with computer vision, helping tasks that involved deriving information from digital images, videos and the like. In common practice of computer vision modeling, transfer learning is used to train convolutional neural networks on a certain task, and then adapted/"fine-tuned" on a new task. With this technique, the network can make use of the knowledge it learned from the original task. With transfer learning, the model is split into a body and head, where the body initializes and learns the features and states of whatever source task/domain it was pretrained on. The head then is the part we get to change and fine-tune to our needs depending on the task we want to perform. In computer vision, *pretraining* starts with training the models on large-scale datasets like Image-Net [14], containing millions of images to learn from. The purpose of this is to teach of computer vision model basic image features - colors, shapes, edges, etc. Once pretrained these models are fine-tuned on a downstream task, which could be anything. Some examples include image classification and object detection. Because the models are already pretrained to understand certain aspects of images, they can operate this fine-tuning step with a small amount of labeled examples. Hence, we are "transferring" the skill set obtained from one task to perform another task. Transfer learning has become the norm due to the fact that these fine-tuned models generally perform better than supervised models that you would train from scratch.

6.3 Transfer Learning in NLP

On the other hand, NLP did not have a clear pretraining process for a long time. The main issue was that NLP applications required not only a large amount of data, but enough of it that was *labeled* if it wanted to perform well. This interested many researchers to figure out a way to make transfer learning viable for NLP. A group of researchers at OpenAI paved the way by introducing a approach where they would use features as a result of unsupervised pretraining to perform a sentiment classification task, with very strong results. [15] Furthermore, it is hard to talk about transfer learning without mentioning ULMFiT, which now serves as the base framework for NLP training tasks. [16] Finally, the need for large amounts of labeled text data specific to our task was no longer necessary. Many describe it as the final "missing puzzle piece" to make transformers the new standard - and some of their performances over older techniques like LSTMs are shocking.

In our case of utilizing transfer learning, the downstream task of course is sentiment analysis. As an example as of how this works, we can take a look at how BERT would operate for a sentiment classification task. In terms of pretraining, we give the model an initial objective of language modeling (predicting the next word in a sentence based off of the previous one). This way, we don't need a bunch of labeled data - specifically, BERT uses the corpora of BookCorpus and the entirety of English Wikipedia. Once the model is trained on this data, the model has to be adapted to the domain. In our study, it would be the dataset of all of our Ukraine vs. Russia tweets. We still aren't predicting polarity yet, as in this state the model continues to try to predict the next word.

The final step is indeed the fine-tuning, which is where the classification layer for sentiment analysis is implemented. The nice part about this is that we can implement custom steps for each model and task. The unwieldy aspect is that the logic is different and isn't always standardized depending on the creator of the model. Luckily, we have an interface called Huggingface that has created a unified API filled with thousands of different transformer models all adapted for different tasks under a common framework to make NLP that much easier. In addition to the API, Huggingface offers a very beginner-friendly course for doing all sort of NLP tasks, and many of the results were obtained using the pipelines, trainers, and objects they provide. [17]

6.4 Transformer Architecture with Sentiment Analysis

Below we can look at a diagram that is a simplified version of how a transformer works for a sentiment analysis task. Similar to older models, we still tokenize data and break things down word by word. We are going to specifically use iterations of BERT, which does its encoding bidirectionally and uses a special type of modeling called *masked language modeling* (MLM). Instead of trying to predict the next word in pretraining/adaptation, the task is to predict randomly masked words in the text. As an example, "I went to the `[MASK]` to go meet `[MASK]`". The model is trained to predict the most likely candidate for each masked word. Other than that the architecture is similar to what is described in Sections 6.1 and 6.2 which can be seen in the diagram below:

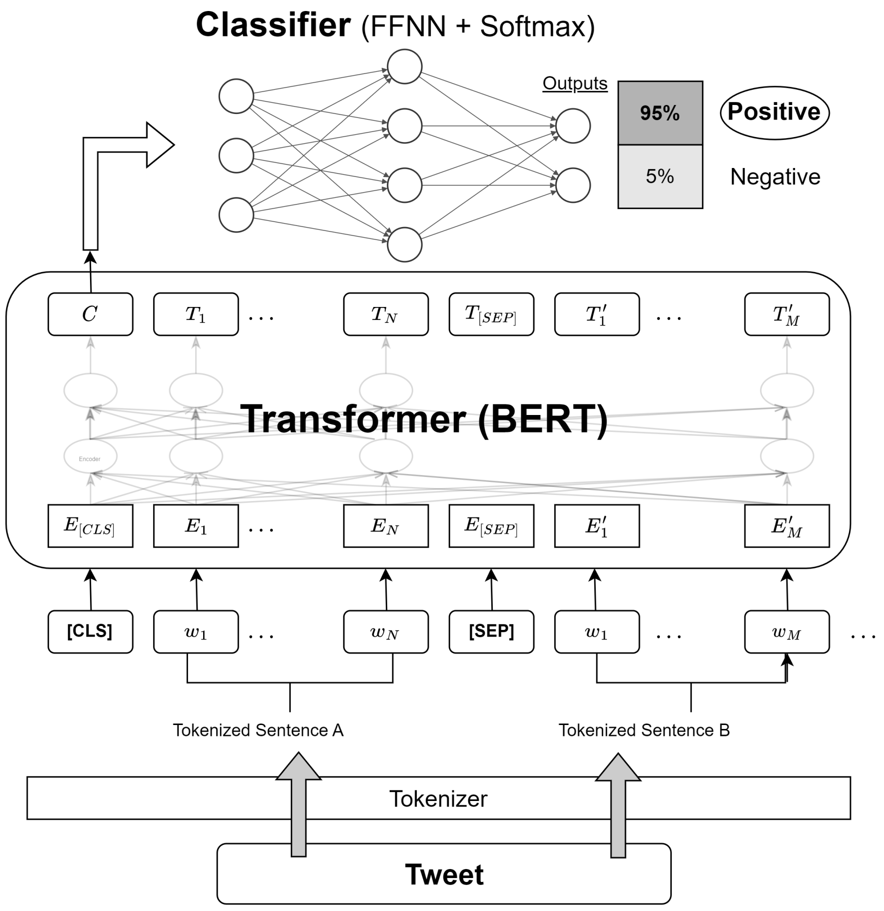

Figure 2: Architecture of how a Tweet is processed through transformer architecture and outputted for sentiment classification

BERT specifically only requires the encoder part of the transformer architecture. Observe how the output class is spat out into a FFNN. A simple softmax [18] function is then able to output the probability of whether it thinks a tweet is positive or negative given the pre-training, adaptation, and fine-tuning process to evaluate the context and significance of words thanks to self-attention.

6.5 Differences in Preprocessing Methods

Remember when we had to do all of that stemming, lemmatization, and removal of junk in attempt to reduce the vocabulary size of our bag of words? Since transformers are trained off of a corpus of data that includes punctuation, emojis, etc. we actually have to do little to no preprocessing. We can basically throw the tweets into the model raw, minus handles with an "@", trailing whitespace, and links. Each model we discuss tokenizes the sentences and words in each tweet a little differently, so we will save the explanation of that when we discuss the differences in each model that was tested. Other than that, we do the same process of removing that VADER-predicted neutral class because it doesn't add anything interesting to our analysis.

7 "New School" Classification: Transformers

We will go over all of the transformer models we tried to implement and train for the task of sentiment analysis. We will use the pipelines provided in Huggingface's `transformers` Python library for all of the pre-training, adaptation, and fine-tuning of the models we use. We are going to through a couple of transformers that have rose to popularity due to their ability to perform well. Keep in mind that while all of these have slightly different methodologies, the main point is to evaluate how they collectively perform better than old school NLP methods. Note that all of these are pre-trained transformers, all trained some sort of large data set in order to learn the general structures and patterns of the language which will help us in our downstream task for predicting the sentiment of our Twitter data.

7.1 BERT

We have already talked about how BERT works a little bit architecturally. Recall that BERT stands for “Bidirectional Encoder Representations from Transformers”. The important words here are *bidirectional* and *encoder*. Like mentioned before, in traditional NLP, text is processed sequentially (like with RNNs and LSTM), where models can only see the words that come before or after a given word. With BERT, every token is linked to one another in both directions, so context is derived from the whole sequence. This tremendously improves the accuracy of the model. And of course, BERT only contains the encoder part of the transformer, like many of the models that will follow as well just as they often are the standard for performing a sentiment classification task after.

Architecturally, BERT consists of multiple layers, each of which is made up of multiple “attention heads” that focus on different parts of the input text. There are two main model types:

- `bert-base`: 12 transformer blocks, 12 attention heads, 768 hidden layer size
- `bert-large`: 24 transformer blocks, 16 attention heads, 1024 hidden layer size

For most of our models, we will use the “base” version of them as the large ones take a longer to train and fine-tune and are a bit overkill for our purposes. The hidden layers have the same purpose as any deep neural network, in which data can be transformed and contextualized based off of each of them. You may also notice that we use `bert-base-uncased` from Huggingface. The only difference is that in the uncased version, all proper nouns and words with capital letters are lowercased. Usually the difference between uncased and cased isn’t too noticeable, but it is more common practice to just lowercase everything. We already talked about MLM and next sentence prediction (NSP) that is used for training BERT. There have been a lot of variations of BERT, however, that have been created with the goal of making it more efficient in all aspects.

One thing you will see that is different between these models is how they are tokenized, and while we won’t explicitly go into each one, we will briefly touch on how BERT’s WordPiece tokenizer works as it uses a much more clever approach compared to what older models like BOW do. Words are split into either its full form, or into word pieces when we have things like prefixes, suffixes, and gerunds that add to a root word.

Word	**Token(s)**
surf	['surf']
surfing	['surf', '##ing']
surfboarding	['surf', '##board', '##ing']
surfboard	['surf', '##board']
snowboard	['snow', '##board']
snowboarding	['snow', '##board', '##ing']
snow	['snow']
snowing	['snow', '##ing']

Table 5: Examples of how BERT's tokenizer processes words

This is why we choose to not do much preprocessing like stemming. Transformer tokenizers actually use every part of every word to its advantage. It is not a hindrance like it would be for a BOW. This is just one of the many things that makes these transformer models a lot better than old school methods. All of our Ukraine data is tokenized in this way - the only annoying thing is that we have to tokenize it differently to adhere to each model, so we end up splitting our data 8 times (4 for each transformer we test, and 2 more each of those for train/test split for model evaluation).

7.2 RoBERTa

In 2019 came along a study that revealed that how BERT's performance could be further improved. RoBERTa, which stands for Robustly Optimized BERT Pretraining Approach, was developed by researchers at Facebook AI with the goal of modifying the pretraining scheme to make it more optimized. [19] RoBERTa uses a larger training dataset and more training steps than BERT, which makes it a more powerful language model. Additionally, RoBERTa uses a different training objective than BERT by dropping the NSP task, which helps it to better capture the nuances of language and improve performance on natural language processing tasks.

Another difference between BERT and RoBERTa is that RoBERTa is designed to be more robust to the effects of hyperparameter tuning. BERT can be fine-tuned for a specific task by adjusting the values of certain hyperparameters, but this process can be time-consuming and requires a lot of experimentation to get right. RoBERTa, on the other hand, is less sensitive to the effects of hyperparameter tuning, which makes it easier to use and more consistently effective across a range of tasks. We choose `roberta-base` for the same reason we choose the base version of BERT. RoBERTa has the same architecture as BERT, but uses a byte-level "byte-pair-encoding" (BPE) tokenizer (the same as GPT-2). While we will not get into the specifics of how it works, the "Summary of the tokenizers" chapter of the Huggingface transformers course [17] does a great job at breaking down the tokenization algorithms and what differs from BERT's WordPiece tokenizer.

7.3 ALBERT

The ALBERT model was introduced around a similar time as RoBERTa, coming with three main changes to the encoder architecture. First of all, the token embedding matrix is decoupled from the hidden dimension matrix. This allows the tokenization process to save on parameters in cases when there is a very large vocabulary size. The second change is that the layers are repeating, and thus share the same parameters throughout. The third change - and similar to RoBERTa - the NSP pretraining objective is dropped and replaced with a task where the model needs to predict whether or not the order of two consecutive sentences was swapped, instead of predicting whether they relate to each other at all. The goal of ALBERT really is parameter reduction, and according to Z. Lan et al., it runs a lot faster on benchmark tests than BERT does. [20]

7.4 ELECTRA

The interesting thing about ELECTRA as it has no architectural changes from BERT. All of the differences in ELECTRA come in the pretraining objective, and specifically try to fix some of the caveats with the masked language modeling (MLM) task. The limitation of MLM is that at each training step only the masked tokens are updated, while the original input tokens are not. ELECTRA uses a two-model approach: the first one operating like a standard MLM in predicting masked words and tokens. The second model is some thing called the *discriminator*, which has the goal of predicting which of the token from the first model's output were the ones that were originally masked from the getgo. For every token, the discriminator is performing it's own binary classification for every word that is processed, which makes training much more efficient. In Huggingface's Python library, we use Google Research's implementation, with an object name of `google/electra-base-discriminator`. Let's see how big of a time difference this makes.

7.5 Model Comparison & Evaluation

It is finally time to see how these models actually perform when fine-tuned to our Ukraine data and subsequently evaluated on its ability to perform sentiment classification and analysis. All of our models were trained for 4 epochs to give equal comparison of speed. 4 epochs was enough run time to train to a point where training loss was negligible (< 0.1). The models also shared the same batch size, AdamW optimizer, and number of warmup steps to get an equal comparison. Note that we could try to hyperparameter tune each model to get it perform the best, but for this case we care more about the comparison to the old school methods. The data and models are trained and evaluated in the PyTorch implementation. However, there options to accomplish the same tasks using Tensorflow. When comparing accuracy and time it takes to fine-tune our models, this is what we get:

Model	**Accuracy on Test Data**	**Fine-Tuning Time (4 epochs)**
BOW + Naive Bayes	80.5%	6 sec
BERT	88.2%	14 min, 29 sec
RoBERTa	88.2%	14 min, 10 sec
ALBERT	86.7%	13 min, 18 sec
ELECTRA	87.4%	12 min, 34 sec

From the results, we clearly see the negative correlation between accuracy and training time. Because all of these models are based off of the BERT architecture, this gives us a good sense in how the different pre-training methods really did effect the speed of fine-tuning our model in the end. The truth is, that this is only a small pool of transformer models, and more and more continue to be released all of the time due to how new they are. 5 years from now, for all we know, these could even become obsolete. With some more hyperparameter tuning and training time, these models could reach 90% accuracy in predicting the correct sentiment of our model (this can be done just by implementing the `large` version of these models if we want to sacrifice training speed). Of course, the elephant in the room is that a BOW implementation takes virtually no time to train (because it is just calculating bayesian probabilities, ie. just matrix multiplication), but keep in mind that this is a model that took a lot of tuning and preprocessing (and some trial & error) to even reach the 80% threshold. If we don't run any of the preprocessing methods like stemming or removing stop words, the Multinomial Naive Bayes classifier outputs an accuracy in the range of $65-70\%$. All of our transformers need next to no preprocessing due to their ability to rely on tokenization and especially self-attention for learning context of certain words.

In terms of the best model to choose for a given task, the best answer is that it depends on *your* specific task. If you are trying to replicate the sentiment analysis and classification of tweets related to the Ukraine Crisis, then use a variation of RoBERTa or ELECTRA. However, the bigger point here is that these transformers have completely revolutionized the NLP field, making old school models like BOW and RNNs quite obsolete. The future of NLP for transformers is exciting, and it will only get better as more efficient paradigms for encoders and decoders are discovered. We haven't even addressed things like sentiment analysis in different languages, which is possible with models like the incredibly popular XLM-RoBERTa. There also exists many "distilled" versions of all these models that try to increase the training speed while minimizing the loss in functionality. The world of transformers is evergrowing and here to stay, as these models truly are the future.

8 Conclusion & Future Applications for Sentiment Classification

Natural language processing is used by a variety of organizations to gain real, actionable insight on many aspects of life. NLP allows us to make more accurate predictions about the world around us, in addition to streamlining processes and minimizing the costs of all sort of tasks. This report described the methodology of using Twitter in combination with machine learning as a medium for learning more about the patterns of the emotions of individuals around the globe. We outlined the process for scraping data, as well as acknowledged the problem of dealing with unlabeled data. As we gain more and more data, the question of whether unlabeled data is useful continues to come up within the data science field. Especially with transformers, who inherently don't need a ton of actual training data for a specific NLP task due to the extensive pre-training process that is possible with transfer learning, the once essential need for labeled data truly isn't a problem anymore.

As mentioned previously, all of the transformer models we tried just utilize an encoder simply because it works the best for sentiment analysis. However, there is a whole world of machine translation (not just to human languages by the way, even to programming languages!), chatbots, virtual assistants, search engines, and so much more. Seriously, there is an entire side of transformers that we didn't talk about, being the decoders. Most famously, the GPT family created by OpenAI is the decoder equivalent of the BERT family. Compellingly, as this report was being finalized, OpenAI released a chatbot that is taking social media by storm, and its potential is crazy. In early December 2022, they release a free research preview of a chatbot that was heavily optimized for human diaologue by using Reinforcement Learning with Human Feedback (RLHF). The AI system, called ChatGPT, is fine-tuned on their GPT-3.5 model. ChatGPT can handle even some more complex instructions and requests that has never been seen by a chatbot before. This is very promising for the future of transformers and natural language processing, and an AI that can convincingly generate human-level text and advice is something that we once thought was still a good ways away.

All corresponding code with the full analysis is available as a Jupyter notebook on the Github repo for this project. Code that involves utilization of the Twitter API and `snscrape` has been commented out to prevent further pulling of data. None of this would have been possible without thanking Dr. Eitel Lauría for his guidance throughout the course of the project.

References

[1] Kate Conger, Mike Issac, Ryan Mac, and Tiffany Hsu. Inside elon musk's takeover of twitter - the new york times, Nov 2022.

[2] M Trupthi, Suresh Pabboju, and G Narasimha. Sentiment analysis on twitter using streaming api. In *2017 IEEE 7th International Advance Computing Conference (IACC)*, pages 915–919. IEEE, 2017.

[3] JustAnotherArchivist. snscrape: A social networking service scraper in python, 2022.

[4] Martin F Porter. Snowball: A language for stemming algorithms, 2001.

[5] Clayton Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the international AAAI conference on web and social media*, volume 8, pages 216–225, 2014.

[6] Steven Loria. Textblob documentation. *Release 0.16*, 0, 2020.

[7] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 1(12):2009, 2009.

[8] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1003–1011, 2009.

[9] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[10] Hiroshi Shimodaira. Text classification using naive bayes. *Learning and Data Note*, 7:1–9, 2014.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[13] Lewis Tunstall, Leandro von Werra, Thomas Wolf, and Aurélien Géron. *Natural language processing with transformers: Building language applications with hugging face*. O'Reilly, 2022.

[14] Imagenet. https://image-net.org/, 2022.

[15] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017.

[16] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[17] Hugging Face. The hugging face course: Transformers, 2022. https://huggingface.co/course, 2022. [Online].

[18] Softmax function, Nov 2022.

[19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[20] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.