## **Part 1: Data**

1. **Aggregation**

I aggregated the data from Twitter. Originally my plan was to use Twitter's Developer Program in order to scrape the tweets online but no matter what I did and how long I was trouble shooting, I did not manage to get the Twitter API to work on my code. This led me to using the website "Vicinitas.io”, a Twitter scraping site that uses its own Twitter API to retrieve data. In fact, this worked slightly to my favor as well, as default Twitter Developer Program members can only pull a grand total of 1,500 tweets per month, while Vicinitas on the other had pulled 2,000 tweets per search and there was not any limit or recharge time. This alone meant my training set of 1,000 tweets had turned into a training set that contained nearly 8,000 tweets, an 800% increase!

From Vicinitas, I gathered the tweets of 6 politicians, with them being Joe Biden, Vladimir Putin, Justin Trudeau, Narendra Modi, Volodymyr Zelenskyy, and Xi Jinping. I pulled 2,000 tweets for each politician (Or whatever the max available was, for Xi Jinping it went only to roughly 1,500 tweets) by searching for them, using their respective "Hashtag" on Twitter. Once the tweets were scraped, Vicinitas placed them in a excel file for me and I converted the file over to a csv file so it could be analyzed further in my coding development environment (Jupyter Labs in this case).

1. **Preparation**

Going more in depth about the data I got, as I mentioned the data was from the website Vicinitas. It gave me the output and then also allowed us to transfer the output to an excel file. Once I transferred the data to excel, I realized that there was alot of extra information in the excel file, data that pertained to how many retweets, likes, mentions a tweet got and so on. In the case of our NLP project where we are analyzing the tweets and the emotional/polarity values within, the one true metric that was important to grab was the actual text within a tweet. This led me to modify the initial csv files only to contain 4 rows of data: The user who tweeted, text of the tweet (Or the tweet itself in other words), language in which the tweet was written (In case it was a language that Textblob, the python library I used to do sentimental analysis, didn't have in its repertoire), and type of tweet (If it was a retweet(repost), an original tweet, or a reply to a tweet). After this, I had a clean data set that contained the necessary information I needed and moved it over to Python for even more refinement. The first thing I did was realize was that there were a high number of retweets in the data. I did a quick analysis, and some tweets were repeated upwards of 30-40 times in the set of 2000 tweets per politician! This meant that multiple tweets that contained the exact same text existed in my pool of training data. Because of that I made a quick function that made new csv files which only contained tweets and replies, thus ensuring that there was no instance of reoccurring text within the actual tweets and as a result altering the overall polarity scorings. For the rest of the project, I end up using those 6 altered csv files. The second thing I did was create a new data set the contained the top 100 words along with their polarity scores for our training data. The 4 politicians I decided to use for training data were Joe Biden, Vladimir Putin, Xi Jinping, and Justin Trudeau. The top 100 words were pulled from a separate sentiment analysis I did on the politicians’ own tweets. After the analysis I made code that derived the top resulting words in that analysis and their respective polarity scorings. When I initially pulled the top 100 words, I ran into a situation where the top pulled words were almost all words that contained no sentimental/polarity value, meaning that their scores were just a flat 0.0. This led me to further adjust the code to make it where it pulled the top 100 words that occur where polarity ISN'T equal to 0, so that way I can get data on words that do contain a level of polarity and use that for the testing. During the training and test phases, when it came to missing values, if a value was missing, I just dropped it from the set so it wouldn't interfere with the other tweets. In my case however, I didn't really have any missing data, so it wasn't a problem. What was a problem was the number of redundancies I had. In my top 100 list, during the testing phase I realized I had all this information in separate csv files, so I decided to concat them into in database, but this resulted in top words repeating within that list database causing an error when it came time to test. I ended up taking a similar approach as I did with missing data and just made it to where I skipped over a repeating word so as to not alter the dataset too much while retaining the top 100 words.

1. **Analysis**

My project was based on gathering a set of tweets and using textblob to do sentiment analysis on those set of tweets, pull the polarity/emotional values of said tweets, and based off the results I get, use those results to train a machine and do a sentimental analysis but this time not using textblob but rather using the information derived from the set of tweets to do a sentimental/polarity analysis on a test set and then compare the results to the textblob results to see how accurate the machine was in terms of identifying the emotional level of a tweet. In order to begin my project, I first had to scrape data from twitter, so I had a set of tweets to work with. I needed a dataset that contained emotional statements, originally, I was going to do the project based on video games but when I would search Twitter for any game, what came up was videos, fan art, and other types of tweets that didn't really have alot of text but rather were a form of media. This led me to choose politicians as the hashtag to search for because they contained alot of highly emotional text-based responses from Twitter users across the world. I scraped the tweets and transferred them over to an excel file, did further modifications to reduce it only to the data I needed to train, converted to csv and then I sent them over to Python for further analysis and preparation. Once in Python, I cleaned the data one more time for all politicians and removed retweets from the pool of tweets in order to insure I wasn't using multiple of the same tweet/text to train. This reduced most of the datasets from 2000 tweets to around 500-600, but the result was more pristine data to use. After this step I did sentimental analysis on the 6 politicians’ overall tweets and got a plethora of information. I received the total polarity value for each separate tweet, the average polarity, and the most important of the data, a list of the top 100 recurring words with non-zero polarity and their respective polarity scoring for each of the 6 politicians (Although I only needed the 4 I was going to use for training). I then transferred that database and turned it into a csv so I could further manipulate if I needed to down the line. Essentially at this point I was ready to move onwards to the training and testing, with the top 100 lists of Biden, Putin, Zelenskyy, and Jinping and the train, and then comparing those to the overall list of tweets for Modi and Trudeau from the modified csv that contained no retweets.

1. **Visualization**

Below are snippets of the top 25 words along with their polarity ratings for the 4 training subjects (Biden, Putin, Jinping, and Zelenskyy) I received and the polarities of the textblob polarities of the 2 training subjects (Trudeau and Modi)

**Joe Biden Vladimir Putin**

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

**Xi Jinping Volodymyr Zelenskyy**

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

**Narendra Modi TextBlob Polarity**



**Justin Trudeau TextBlob Polarity**



## **Part 2: Machine Learning**

1. **Brief Summary**

The problem I was trying to solve was to see if I can use a small dataset that derives sentiment value from a python library, and then without using that library again, see if I can replicate a sentiment analysis on a dataset via machine learning. I will determine if comments on social media about a topic (In this case politicians) are positive, negative, or neutral. This is a classification task due to the fact that I am categorizing tweets based on their respective polarity scoring (Neutral, Positive, or Negative)

1. **Existing Solutions**

The biggest problem I encountered was actually the need to organize my data first rather than any machine learning aspect of it. Prior to this class I normally never engaged in precleaning of data, and what I mean by that is that I typically never cleaned data from within a csv file or make additional modifications to it, I tried to work with the file on hand as it was. Methodology I learned from the first set of assignments from this class showed me the importance of having nice, clean data and how having a dataset that fits your needs perfectly instead of having to constantly jump around and make modifications later down the line is key. This led me to have an extensive data cleaning segment for this project just so I can ensure that the data I had wouldn't produce errors later on and was easy to understand once it did go into the training aspect of machine learning. It was similar to how we had to sort the messy Hogwarts data and then eventually moved that data over for machine learning without any issues, that was what I was trying to replicate for this project the level of ease in transferring the data over. I also used sklearn for this project, which was the same machine learning we used in assignment 4 to prepare the data (In addition to tensorflow but I didn't use that for this specific project).

1. **Chosen Approach**

The machine learning approach I took for this assignment was to use Support Vector Regression (SVR) based on the scikit-learn library. The reason I went with this approach is because it can handle non-linearly separable data by transforming input data into a higher dimensional space. SVR is a variant of the Support Vector Machine (SVM) algorithm typically used in regression analysis, and I bring this up because SVM's are very efficient at handling high-dimensional data which is that text data (The primary data type I used for this assignment) typically is. I used TfidVectorizer in tandem with this, as I used that to convert text data into numerical vectors for further analysis within my project.

1. **Evaluation**

The model performed greater than I thought it was going to. The overall accuracy of my results was 89.4%, meaning my model was good at correctly identifying and labeling a tweet into the categories of negative, neutral, and positive. It performed wonderfully at identifying neutral tweets with a precision of 92% and recall of 95%, but it performed less so for positive and negative tweets, at precision 73% and recall 59% for positive tweets and precision 65% and recall 60% for negative tweets. I’m assuming that positive tweets got identified more successfully than negative tweets because positive had nearly double the support that negative had in this project. A key metric I wanted to compare was the polarities between textblobs results and the machine learning results. Below are snippets of the results.



As we can see, they are very close! off by ~.19 for Modi and ~.12 for Trudeau. This was the original metric I was going to compare but here is a Classification Report I ended up creating (And where I pulled the values in my beginning paragraph).

A picture containing text, screenshot, font, receipt

Description automatically generated

There is one primary key method to improve my results, and that is simply training it with more data. The machine learning aspect of the code only had 47 samples of negative words, 88 positive, and a huge 707 for neutral meaning that of course its going to correctly identify neutral 90%, it has such a huge set to train from in comparison to positive and negative tweets. Increasing the amount of positive and negative words to use in the training data would give our machine learning more data to derive its analysis on, thus leading to more matches on whether or not the overall sentiment of a set of tweets is positive, negative, or neutral. Simply put, if I had more time, that time would be invested in adding more and more politicians and their series of tweets, getting top occurring words and its polarity value, and using that to train the model even further and produce more accurate results.

**References**

[1] Vicinitas. (n.d.). Download Search Tweets - Free Tools. Retrieved May 5-6, 2023, from <https://www.vicinitas.io/free-tools/download-search-tweets>

^I did 6 different inquiries here, pulling data for the politicians I used. Their tweets stem from May 5th-6th and all credit for the tweets go to the respective Twitter users who posted said tweet (For more you can view the original csv files I attached alongside to see their usernames for credit)

[2] DataCamp. (n.d.). Text Analytics for Beginners using NLTK. Retrieved May 5, 2023, from <https://www.datacamp.com/tutorial/text-analytics-beginners-nltk>

[3] scikit-learn developers. (n.d.). sklearn.feature\_extraction.text.TfidfVectorizer — scikit-learn 0.24.2 documentation. Retrieved May 6, 2023, from <https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html>

[4] scikit-learn developers. (n.d.). sklearn.svm.SVR — scikit-learn 0.24.2 documentation. Retrieved May 7, 2023, from <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html>

[5] EliteDataScience. (n.d.). Python Machine Learning Tutorial, Scikit-Learn: Wine Snob Edition. Retrieved May 6, 2023, from <https://elitedatascience.com/python-machine-learning-tutorial-scikit-learn>

[6] TextBlob Developers. (n.d.). TextBlob: Simplified Text Processing — TextBlob 0.15.3 documentation. Retrieved May 7, 2023, from <https://textblob.readthedocs.io/en/dev/>