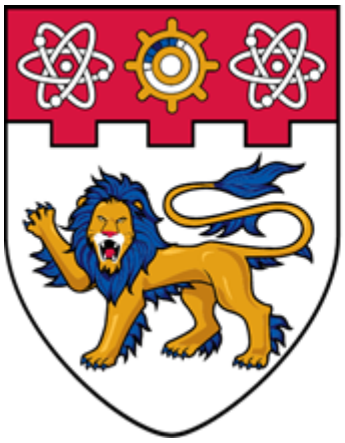

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4034: Information Retrieval Project

Sentiment Analysis of Various Countries on 2020 US Presidential Elections

Group 17:

Das Atrik (U1823950C)

Padhi Abhinandan (U1823860D)

Chandrasekhar Aditya (U1923951A)

Rajuravi Vishal Raj (U1822268B)

Gunturi Kushal Sai (U1923232F)

Jose Jeswin (U1822068H)

School of Computer Science and Engineering (SCSE)

Contents

1 Introduction

November 2020 witnessed a remarkable incident that uprooted the political landscape of the world which was the 59th US Presidential Elections. Unlike past elections held in the US, this one was special because one, it was held during the advent of a worldwide pandemic which meant that common post-electoral activities such as campaigning, political speeches and debates had to be done through online means and the spread of information was mainly through Twitter which was a popular social medium for political news. Moreover, the two candidates who were contesting for the presidency - Donald Trump representing the Republicans and Joe Biden representing the Democrats - had highly contradictory perspectives which led to highly polarized support on both sides. People either loved or hated them.

Thus, we decided to build an information retrieval and classification system which represents whether a country is Pro-Biden or Pro-Trump on a world map based on the sentiments of their tweets on the topic of 2020 US Presidential Elections. The gradient of a country's color ranging between red and blue would show its sentiments towards Trump (Pro-Trump) and Biden (Pro-Biden) respectively.

In order to build this system, first, we scraped the tweets for several countries that we personally judged to have polarizing views on the topic at hand using SNScrape. Then, we used Solr to index and store the data. Finally, we used a BERT-based deep learning model to classify the data into 3 classes for the corresponding tweets - Pro-Trump, Pro-Biden, and Neutral - and then represent the countries on the world map. This report will elaborate on these topics in more detail.

2 Crawling

For this project, our group created an Information Retrieval system based on tweets (short posts) crawled from Twitter about the US presidential elections 2020 from people of various countries.

2.1 How we crawled the corpus and stored it

To scrape the data required for our analysis, we crawled the corpus using SNScrape, an open-source social media scraper. It works on any environment containing Python 3.8 and above. We decided to use this tool as other popular scraping tools such as Tweepy, which uses the Twitter API directly, requires a long set-up such as applying for a developer's account, and getting a token and a secret key. Also, using Tweepy only allows up to 200 tweets to be scraped per request. SNScrape bypasses all these limitations which makes scraping a much easier process.

We explored a wide range of hashtags and narrowed it down to the following 41 relevant hashtags about Donald Trump, Joe Biden, and the US Elections. Note that we considered both "Pro-" and "Anti-" hashtags for Trump/Biden to find more relevant tweets.

```
# Mentions of Trump: 7
"#Trump", "#trump", "#Trump2020", "#DonaldTrump", "DonaldJTrump", "Donald Trump", "Trump"

# Pro-Trump: 8
"#MAGA", "#PresidentTrump", "VoteRed", '#MakeAmericaGreatAgain', '#TeamTrump', '#VoteTrump', '#DrainTheSwamp',  "#MyPresident",

# Anti-Trump: 7
"#VoteTrumpOut", "#DumpTrump", '#TrumpIsPathetic', '#TrumpCorruption', '#VoteHimOut', '#YoureFiredTrump', '#TrumpHasToGo',

# Mentions of Biden: 6
"#Biden", "#biden", "#Biden2020", "Joe Biden", "#JoeBiden", "Biden",

# Pro-Biden: 5
"#VoteBlueToSaveAmerica", "#bluewave2020", "VoteBlue", '#TeamBiden', '#joementum',

# Anti-Biden: 7
"Sleepy Joe", "#SleepyJoe", "HidenBiden", "#CreepyJoeBiden", "#NeverBiden", "#BidenUkraineScandal", '#HunterBiden',

# Miscellaneous: 1
"#USElections"
```

Fig 1: All hashtags used for scraping

We had also decided to scrape tweets from 23 countries that are either geopolitically significant powers in the world, or have vested interests in the results of this US election. The following is the list of countries we used:

```
["Canada", "Mexico", "UK", "India", "China", "Russia",
 "Ukraine", "Brazil", "Iran", "Israel", "Saudi Arabia", "Singapore"
 "Taiwan", "Japan", "South Korea", "Australia", "Philippines", "Indonesia",
 "Afghanistan", "Germany", "France", "Kenya", "Nigeria"]
```

Fig 2: All countries used for scraping

Twitter has made it relatively difficult to scrape tweets by geography as most tweets do not include the user's location information (geo-tags). Instead, we had to search for tweets within a radius of populous cities within each country in our list. This meant we had to change the radius for each of the cities to prevent the radius from crossing into other countries. For small countries the radius was generally kept below 50 km, whereas for larger countries we increased it to up to 1000 km. The scraping proved especially problematic for Canada and Mexico, which saw a great leakage of tweets from the US. After tweaking cities and radius for each of the countries, we were able to achieve a satisfactory corpus of tweets that were divided geographically.

Other constraints added were the period of time the tweets were scraped from. Initially it was kept from 1st November 2020 to 30 November 2020, although if we did not get enough tweets for a particular country during that one month, we increased the period to one month before and after the election month too. The language was restricted to English only and the CSV files were saved in UTF-8 encoded formats so that emoji icons and other ASCII characters were saved correctly.

username	content	date	country	replyCount	retweetCount	likeCount
memo_ramos_	@JoeBide	2020-1	Mexico	0	0	0
VicoZanetti	Premio Fa Biden wo	2020-1	Ukraine	0	0	0
ambeecious	@SJosepl	2020-1	India	0	0	4
erikgeddes	Get him ir	2020-1	UK	1	0	3
TheBeardedWit	Why don't	2020-1	UK	0	0	2
PerryCavalari	Throwbac	2020-1	South Ko	1	0	5

Fig 3: Format of scraped data

Note that only some of the columns were used from the raw data provided by SNScrape. The above screenshot displays a part of the data in one of the scraped CSVs (with the required columns). The field names for each column and their respective descriptions are provided in the table below.

Field Name	Description
username	Contains the username of the user that posted the tweet
content	Contains the actual tweet
date	Contains the date on which the tweet was posted
country	Contains the country where the user lives in
replyCount	Contains the number of people that commented on the tweet
retweetCount	Contains the number of people that shared that particular tweet
likeCount	Contains the number of people that liked that particular tweet

2.2 What kind of information users might like to retrieve from your crawled corpus?

The purpose of our project is to allow users to obtain an idea of the sentiment of the people of various countries towards the US presidential candidates, i.e., to provide users with an indication of how “Pro-Trump” or “Pro-Biden” various countries seem to be, based on the tweets of people from those countries. Thus, users can retrieve the following information from our crawled corpus of tweets:

1. Tweets relevant to the US presidential elections from various countries around the world, sorted by most relevant first or sorted in reverse chronological order.

2. Tweets that were classified to be in favour of Trump (Pro-Trump) or Biden (Pro-Biden) from each country.

3. The most popular tweets from each country (relevant to the US election), determined by the number of likes, retweets, and replies.

4. A visual representation of the sentiment of countries towards Trump and Biden in the form of a colored map, based on how strongly a country supports either Trump or Biden.

Types of Possible Queries:

1. Tweets about Joe Biden
2. Tweets about Donald Trump from Russia
3. The recent tweets about Kamala Harris from India
4. The most popular tweets about Democratic Party
5. Tweets about Barack Obama

2.3 The numbers of records, words, and types in the corpus

Using SNScrape, more than 100,000 tweets were initially obtained from Twitter. However, we noticed that the scraped tweets contained a large number of tweets that were either irrelevant, or were otherwise unsuitable for sentiment analysis. For instance, some tweets contained more hashtags than normal text, and some tweets were repeated tweets from the same user. Thus, the scraped tweets were carefully processed before they were used for indexing, querying, and classification - more detail on the pre-processing of tweets is provided in section 4.2 of this report (Data Pre-Processing).

After data pre-processing, there were 38,364 tweets (records) in the corpus, and these included tweets that were either Pro-Trump, Pro-Biden, or Neutral. However, after passing each record in the corpus through our classification / sentiment analysis model, neutral tweets were removed from the corpus as neutral tweets would not contribute to the calculations that determine if a country is Pro-Trump or Pro-Biden. Thus, in the end, the corpus contains approximately 27,000 records.

The exact number of records, words, and unique words in the final corpus is provided in the table below:

Number of Records in the Corpus	27146
Number of Words in the Corpus	511249
Number of Unique Words in the Corpus	35916

3 Indexing and Querying

3.1 Overview

Indexing is an optimal data structure technique that minimizes the disks accessed during retrieval of records from the database. For this project, we used the software Solr, an open source enterprise search platform built by Apache Software Foundation. Being a highly powerful technology, it has numerous features which include real-time indexing, full-text search, spell-check, advanced filtering, geo-spatial search, stemming, tokenization and stop-words removal. The main component in Solr is the Lucene library. Lucene is a full-text search library written in Java. We created an inverted index using Solr to optimize the search process. Inverted index helps optimizing the search process and allows fast full text search, at a cost of increased processing when a document is added.

Solr version 8.8.1 is used to index the .csv which contains the crawled data followed by preprocessing of the data and then sentiment analysis.

```
<field name="content" type="text_general"/>
<field name="country" type="text_general"/>
<field name="date" type="pdates"/>
<field name="likeCount" type="plongs"/>
<field name="replyCount" type="plongs"/>
<field name="retweetCount" type="plongs"/>
<field name="sentiment" type="text_general"/>
<field name="url" type="text_general"/>
<field name="username" type="text_general"/>
<field name="weightedSentiment" type="pfloats"/>
<field name="weightedSentimentNorm" type="pfloats"/>
```

Fig 4: Field Type of Solr Schema

Thereafter, the processed .csv is indexed using the inverted index method and is posted in Solr. Given below is a snippet of the posted data in the Solr Admin interface :


```
{
  "username":["memo_ramos_"],
  "content":["@JoeBiden you suck!"],
  "date":["2020-10-23T02:42:15Z"],
  "country":["Mexico"],
  "replyCount":[0],
  "retweetCount":[0],
  "likeCount":[0],
  "url":["https://twitter.com/memo_ramos_/status/1319469151899930625"],
  "sentiment":["Pro Biden"],
  "weightedSentiment":[1.0],
  "weightedSentimentNorm":[0.0],
  "id":"8eb3c3a8-fc5b-45ce-ae3b-24efc044196c",
  "Unnamed__0":[0],
  "_version_":1696494987012734976},
```


Fig 5: Crawled Data on Solr Admin Interface

3.2 Build a simple web interface for the search engine

The web interface was developed using Flask framework along with HTML, CSS and JavaScript. It is designed to ease searching of Twitter posts about the United States Presidential election 2020. The Solr application and Flask application was bridged using a python module 'pysolr'. Crawling and incremental indexing of new data is not implemented in this project as it relies on the data retrieved during the election period from September 1st, 2020 to December 31st, 2020.

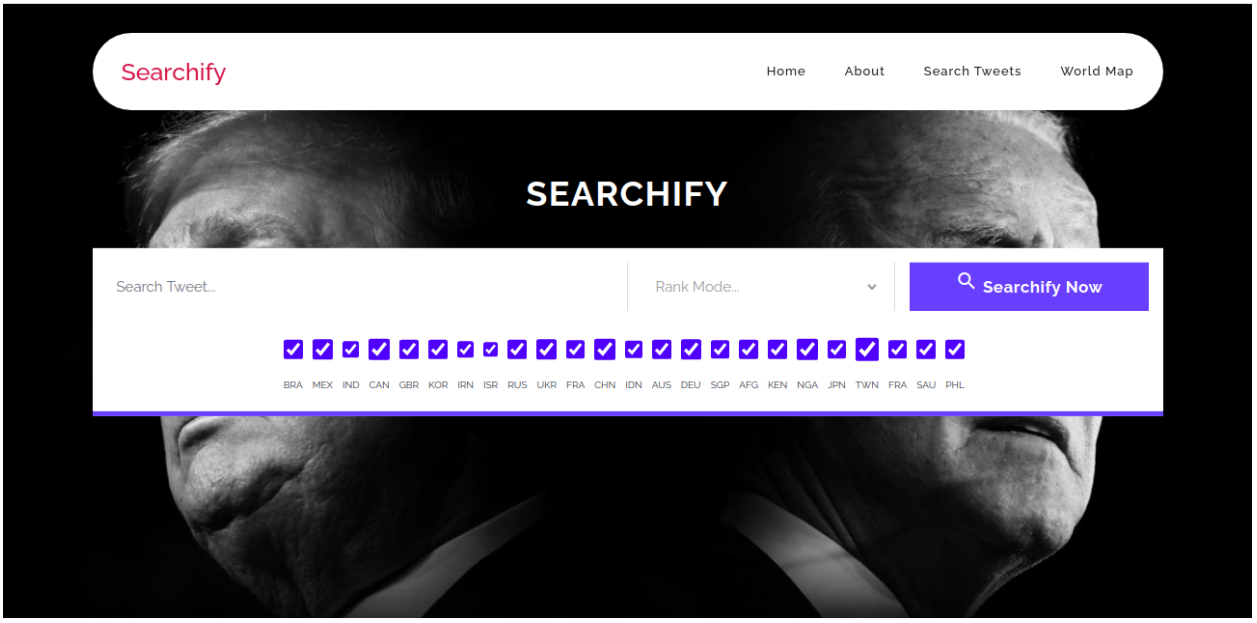

Fig 6: Web Interface for the Search Engine

In order to help ease the process, the user can either search using specific syntax (professional users) or by using words that they wish to find in the tweets. The user can filter the existing tweets by geography. They can also choose a ranking method and the tweets are sorted based on the method chosen by the user. The query result is returned to the user in a user friendly interface with the sentiment (Pro Trump or Pro Biden) marked on it.

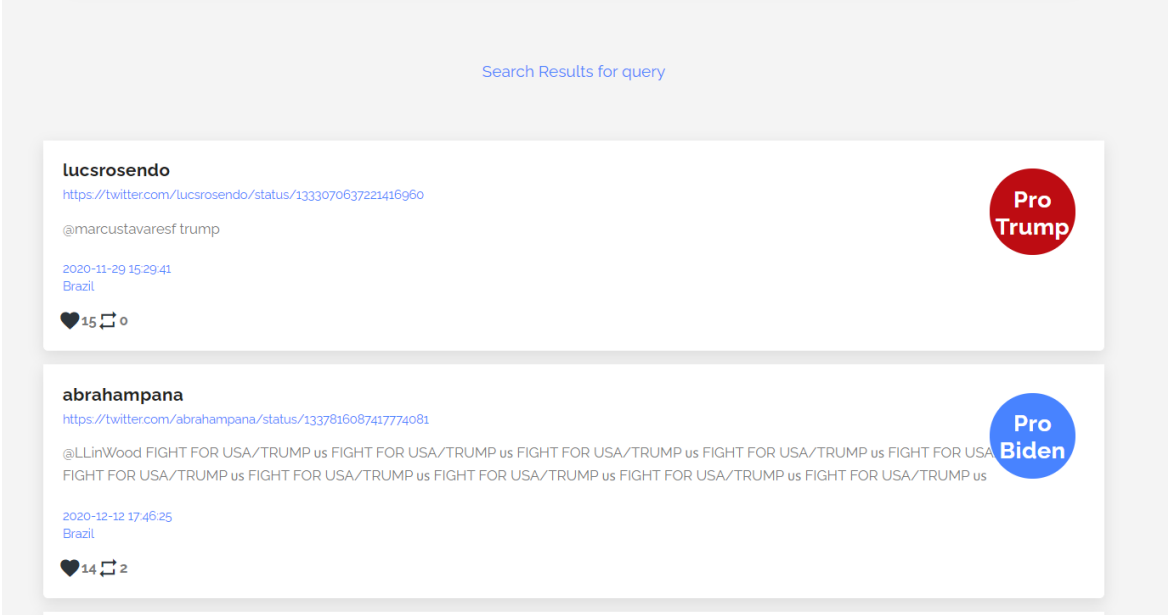

Fig 7: Web Interface for the Search Result

3.3 Write five queries, get their results, and measure the speed of the querying

For querying, we decided to write 5 queries showing various features (not exhaustive) and measure the speed of querying.

Query Type	Query	Filter by Country	Sort by	No of Results	Top Result	QTime (ms)
Word Query	Biden	All	Most Relevant	1768	Biden or trump ,, who's going to win Biden ,biden ,biden I think Joe Biden #ElectionDay2020 #ElectionNight #BidenHarris2020 #Trump2020 #AmericasGreatestMistake	1

Phrase Query	Make America Great Again	All	Most Relevant	33	@realDonaldTrump Make America Great Again!\n#Trump2020	7
Word Query	Trump	Saudi Arabia, Iran, Israel	Most Retweets	70	Hilarious #video of man acting Joe Biden to Donald Trump. #Naijaloveinfo #videos #videography #americans #american #president #presidentelect #joe biden #election2020 @ Nigeria https://t.co/9Yc6V0p1mw	3
Phrase Query	Hunter Biden	South Korea	Most Likes	115	Hannity is still talking about Hunter Biden. Good grief If the GOP is going to normalize post-Trump, cleaning up Fox is the place to start	17
Boolean Query	content : Obama +sentiment:"Pro Trump"	All	Most Relevant	3356	@palkisu That's why I like Trump.. even Obama can never do this.	9

3.4 Explore some innovations for enhancing the indexing and ranking

The application is handling quite a large dataset for review and it is extremely important to ease the users search experience by utilising the variety of enhancement techniques available. For this system, we have implemented the 3 main enhancement techniques which are as follows.

3.4.1 Spell Check

Using words that are spelled wrong is one of the most encountered problems during a search. In order to solve this problem, we designed an interface such that it would provide correctly spelled versions of misspelled words in the form of suggestions to help better serve the user. It is designed to provide query suggestions and corrections. This was done by using the n-grams algorithm to find similar words to a query which are present in the corpus. This algorithm was incorporated into solr for defining it in the solrconfig.xml. The

DirectSolrSpellChecker uses terms in the Solr index which are always up to date with the terms in the index. The n-grams algorithm is performed on these terms to find the closest match and the corresponding suggestion is given to the user.

```
<searchComponent name="spellcheck" class="solr.SpellCheckComponent">
  <str name="queryAnalyzerFieldType">text_general</str>
  <!-- a spellchecker built from a field of the main index -->
  <lst name="spellchecker">
    <str name="name">default</str>
    <str name="field">content</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <!-- the spellcheck distance measure used, the default is the internal levenshtein -->
    <str name="distanceMeasure">internal</str>
    <!-- minimum accuracy needed to be considered a valid spellcheck suggestion -->
    <float name="accuracy">0.5</float>
    <!-- the maximum #edits we consider when enumerating terms: can be 1 or 2 -->
    <int name="maxEdits">2</int>
    <!-- the minimum shared prefix when enumerating terms -->
    <int name="minPrefix">1</int>
    <!-- maximum number of inspections per result. -->
    <int name="maxInspections">5</int>
    <!-- minimum length of a query term to be considered for correction -->
    <int name="minQueryLength">4</int>
    <!-- maximum threshold of documents a query term can appear to be considered for correction -->
    <float name="maxQueryFrequency">0.01</float>
  </lst>
</searchComponent>
```

Fig 8: SpellCheckComponent on solrconfig.xml

After viewing the system's suggestions, users will be able to search using the suggestion or sticking to the default search query. All the suggestions provided are in lowercase since the indexing built is in lowercase. Few examples from the system are shown below :

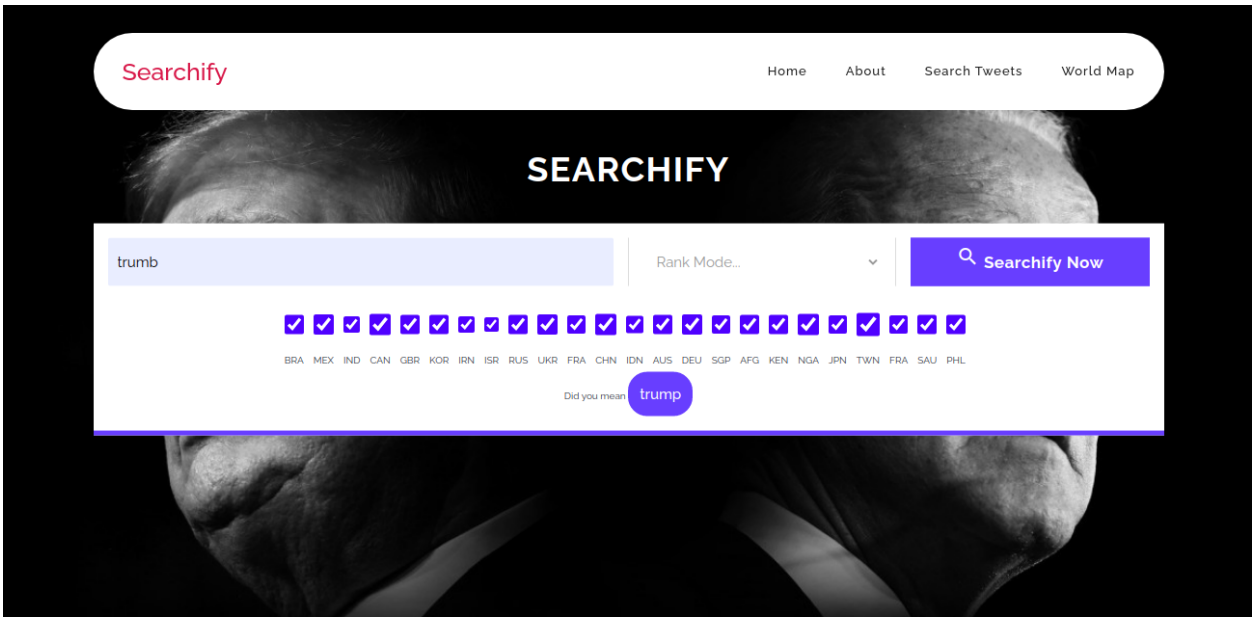

Fig 9: SpellCheck demo 1

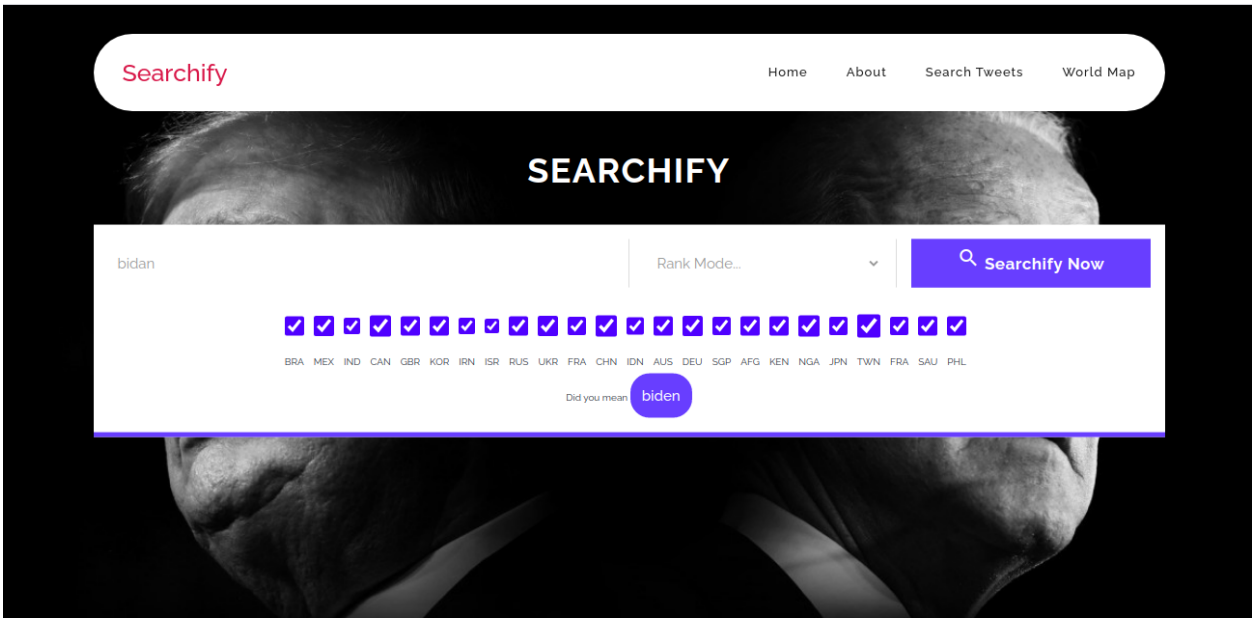

Fig 10: SpellCheck demo 2

3.4.2 Geo-spatial Search

The Geo-spatial search is added in the search engine to enable users to search tweets by multiple countries.The *fq* function in Solr is used to return the results for the given query and checkboxes are added to the search page. Users can filter the search results for 23 countries which are as follows: France, Mexico, China,India, South Korea, Israel, Indonesia, United Kingdom, Australia, Brazil, Russian Federation, Canada, Germany, Singapore, Ukraine, Afghanistan, Kenya, Nigeria, Japan, Taiwan, Iran, Saudi Arabia, Philippines.

Fig 11: Checkboxes for each country

3.4.3 Tweet Sorting

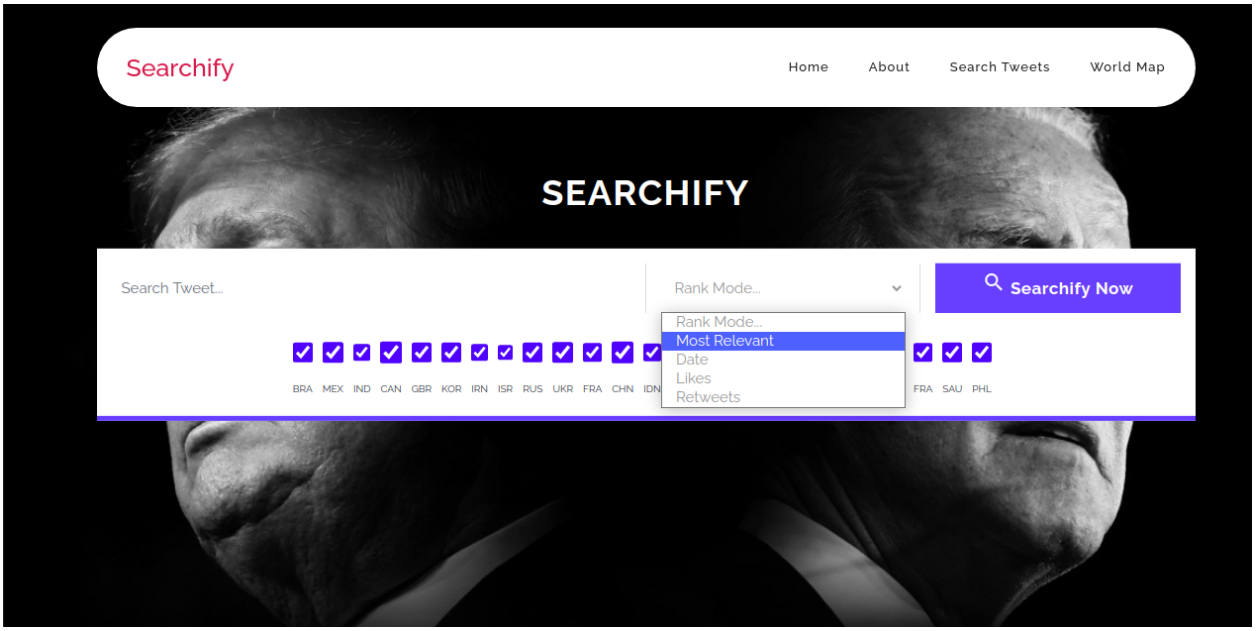

Fig 12: Tweet Sorting

- **Sort by Most Relevant**

  The default setting is returning search results sorted by their relevance score. This method gives the most accurate and appropriate tweets for the search query.

  Example: User searches for tweets about Kamala Harris, the system will automatically sort all the tweets according to its tf-idf score and return the most relevant ones.

```python
@staticmethod
def rank_by_most_relevant_tweets(query, tweets):
    tweet_content = [tweet['content'][0] for tweet in tweets]
    vectorizer = TfidfVectorizer()
    content = tweet_content.copy()
    content.insert(0, query)
    vectors = vectorizer.fit_transform(content)
    feature_names = vectorizer.get_feature_names()
    dense = vectors.todense().tolist()

    query_tfidf = pd.DataFrame([dense[0]], columns=feature_names)
    document_tfidf = pd.DataFrame(dense[1:], columns=feature_names)

    all_scores = list(document_tfidf.dot(query_tfidf.iloc[0]))
    for i in range(len(all_scores)):
        all_scores[i] = [i, all_scores[i]]
    all_scores = np.array(all_scores)
    all_scores = all_scores[all_scores[:, 1].argsort()][::-1]

    ranked_tweets = []
    for score in all_scores:
        tweet = tweets[int(score[0])]
        ranked_tweets.append(tweet)
    return ranked_tweets
```

Fig 13: Code snippet of Most Relevant

- **Sort by Time**

  In this setting, the tweets are returned in reverse chronological order.

  Example: User searches for tweets about Mike Pence, the system will automatically sort all the tweets according to time.

```
@staticmethod
def rank_by_date_tweets(tweets):
    return sorted(tweets, key= lambda x: x['date'][0], reverse=True)
```

Fig 14: Code snippet of Sort By Date

- **Sort by Like Count (Popularity)**

  On Twitter, we often notice that the most popular tweets usually contain a high number of likes. Therefore, we will use likes count to measure the popularity of a tweet.

  Example: User searches for tweets about Donald Trump, the system will automatically sort all the tweets according to the number of likes.

```
@staticmethod
def rank_by_likes_tweets(tweets):
    return sorted(tweets, key= lambda x: x['likeCount'][0], reverse=True)
```

Fig 15: Code snippet of Sort By Likes

- **Sort by Retweet Count (Reach)**

  Most of the trending tweets on Twitter usually contain a high number of retweets. Thus, it becomes a tool to measure the Reach of the user. Moreover, retweet count reveals engagement of tweets among the public. These tweets often contain high interactions by the audience.

  Example: User searches for trending tweets about Joe Biden, the system will automatically sort all the tweets according to the number of retweets.

```
@staticmethod
def rank_by_retweets_tweets(tweets):
    return sorted(tweets, key= lambda x: x['retweetCount'][0], reverse=True)
```

Fig 16: Code snippet of Sort By Retweets

4 Classification

4.1 Motivate the choice of your classification approach

The models we have chosen to use for classification are:

1. Machine Learning
    a. Gaussian Naive Bayes
    b. Random Forest
    c. Extra Trees Classifier
2. Deep Learning
    a. Long Short Term Memory Classifier
    b. BERT Classifier

Machine Learning Models

The models we have chosen to use for classification are *Gaussian Naive Bayes, Random Forest* and *Extra Trees Classifier*.

Gaussian Naive Bayes

The Naive Bayes method is a series of supervised learning algorithms focused on applying Bayes' theorem to every pair of features with the "Naive" assumption of independence. To estimate the necessary parameters, Naive Bayes requires a small amount of training data. When compared to more sophisticated methods, Naive Bayes classifiers can be extremely fast. The decoupling of the class conditional function has a number of implications. Since the class conditional feature distributions are decoupled, each distribution can be calculated as a one-dimensional distribution independently. This, in essence, aids in the alleviation of problems caused by the curse of dimensionality. The assumption for Naive Bayes, on the other hand, is unnecessarily simplistic. The probability outputs should not be taken too seriously, since Naive Bayes is considered to be a poor estimator.

```
F1 score:  0.5280898876404494
Accuracy:  0.41414141414141414
Precision:  0.5949367088607594
Recall:  0.47474747474747475
```

Fig 17: Evaluation metrics for Naive Bayes

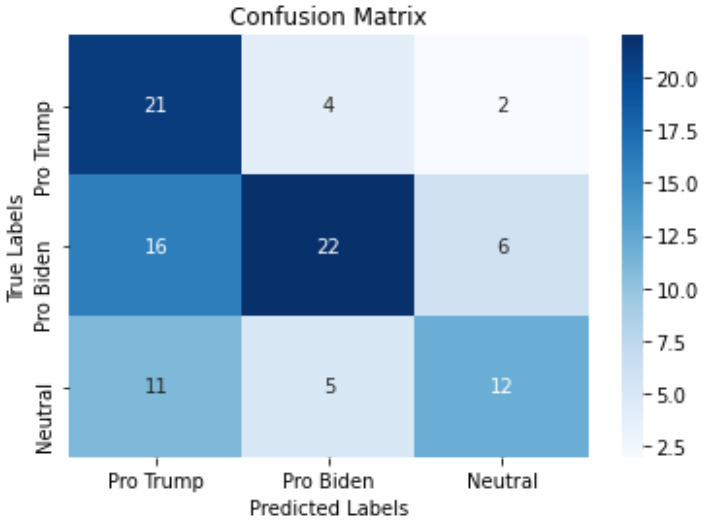

Fig 18: Confusion Metrics for Naive Bayes

Random Forest

The Random Forest (RF) classifiers are suitable for dealing with the high dimensional noisy data in text classification. An RF model comprises a set of decision trees each of which is trained using random subsets of features. Given an instance, the prediction by the RF is obtained via majority voting of the predictions of all the trees in the forest. Random Forest can run efficiently on large dataset. It also offers great accuracy in classifying and estimating. However, the classifications made by random forests are difficult for humans to interpret unlike decision trees. If the data contain groups of correlated features of similar relevance for the output, smaller groups are favored over large groups.

```
F1 score:  0.5897435897435898
Accuracy:  0.46464646464646464
Precision:  0.8070175438596491
Recall:  0.46464646464646464
```

Fig 19: Evaluation metrics for Random Forest

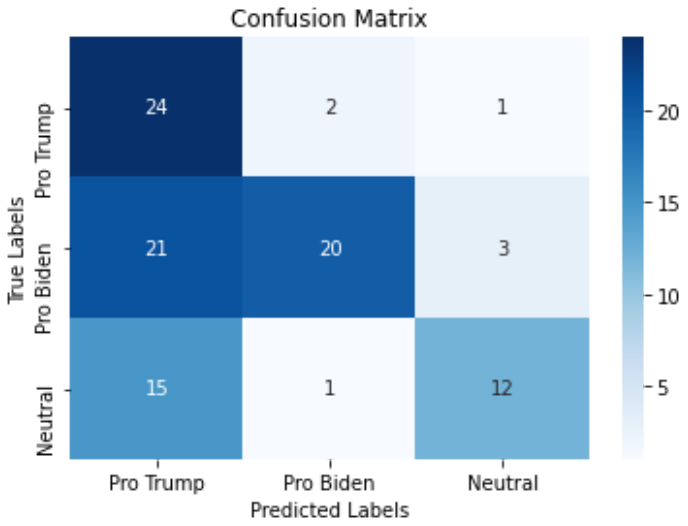

Fig 20: Confusion matrix for Random Forest

Extra Trees Classifier

Extremely Randomized Trees Classifier (Extra Trees Classifier) is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a "forest" to output it's classification result. Extra Trees is like Random Forest, in that it builds multiple trees and splits nodes using random subsets of features, but with two key differences: it does not bootstrap observations (meaning it samples without replacement), and nodes are split on random splits, not best splits. In Extra Trees, randomness doesn't come from bootstrapping of data, but rather comes from the random splits of all observations.

```
F1 score:  0.5903614457831325
Accuracy:  0.494949494949495
Precision:  0.7313432835820896
Recall:  0.494949494949495
```

Fig 21: Evaluation metrics for Extra Trees Classifier

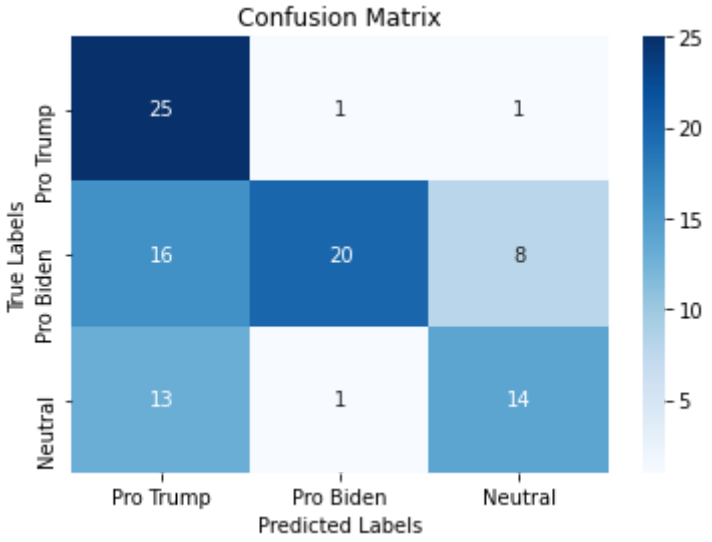

Fig 22: Confusion matrix for Extra Trees Classifier

Deep Learning Models

The models we have chosen to use for classification are *Long Short-Term Memory (LSTM) Classifier,* and *Bi-Directional Encoder Representations from Transformers (BERT)*.

Long Short-Term Memory (LSTM) Classifier

LSTMs are a breed of Recurrent Neural Networks (RNNs) which have the ability to learn long-term dependencies. Unlike normal neural networks which are linear (the data is fed forward and later back propagated to optimize the weights of each layer), an RNN is actually recursive in nature as it contains multiple loops in each layer so that information can persist from one layer to another. An RNN tries to create a Language Model (LM) by reading the corpus of texts from left-to-right and remembering the contexts of each word in relation to its position in the sentence. However, as the sentence gets long and winded, the neural network has difficulties remembering what the first word of the sentence was. This is where LSTMs fix the issue of long-term dependencies.

Each layer of the LSTM network has a sigmoid layer coupled with a pointwise multiplicator which outputs a float between 1 and 0. This layer acts as a "gate" which controls the flow of information to each layer essentially allowing the network to add or remove information according to what it deems to be relevant. The data is passed multiple times through the layers until the LM can be constructed. An LM analyzes a corpus of text and tries to predict

what the next word in a sentence might be based on statistical inferences on the probability of the occurrence of a given sequence of words in a sentence.

The LSTM was the model used for the classification but the methodology used is called *Universal Language Model Fine-Tuning (ULMFiT)*. The core idea of this technique is inductive transfer learning which claims that a model pre-trained on a large database of documents will perform well on related NLP tasks if the same model is fine-tuned on the smaller dataset again.

For this project, we first took a pre-trained LSTM Language Model that was trained on WikiText-103 so that it got an idea of how the intricacies of the English language works. Then we fine-tuned it on our own train dataset. We used novel concepts such as gradual unfreezing of layers where all the layers except the last one are frozen and fine-tuning is done for one epoch from the last layer all the way up till the first layer of the model. This bypasses the issue of fine-tuning all layers at the same time which could result in some of the information being forgotten. We also used dynamic learning rate scaling where the learning rate was increased for shallower levels of the model but decreased for deeper layers of the model which improved the overall performance of the model.

Ultimately, we chose not to use this model because we realized that LSTM required a lot of data to accurately predict the next word of a sentence. We were only using around 2000 labelled data points for training which was not enough. The model was having extremely high (95%+) train accuracies but mediocre test accuracies which meant that the model was memorizing the prediction classes for each tweet instead of generalizing when given a new dataset thus overfitting due to a lack of data.

```
In [374]: interp = ClassificationInterpretation.from_learner(learn_model)

In [375]: interp.plot_confusion_matrix()
```

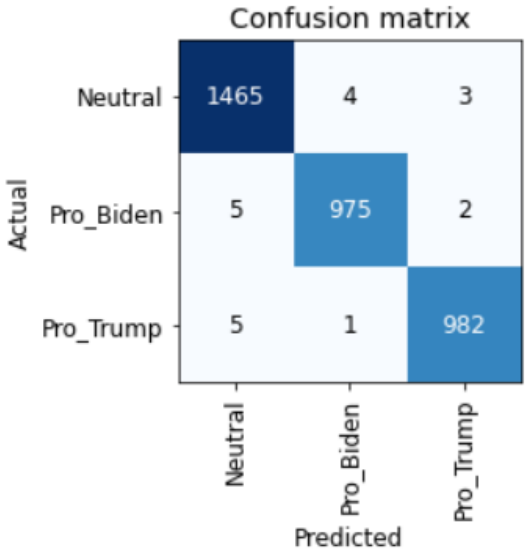

Fig 23: Confusion matrix for LSTM

```
In [369]: # Unfreeze all the layers in the model and train again with a smaller learning rate, the sliced values follows the instructions
learn_model.unfreeze()
learn_model.fit_one_cycle(5, slice(1e-3/(2.6**4),1e-3))
```

epoch	train_loss	valid_loss	accuracy	f1_score	time
0	0.049525	0.034271	0.988960	0.988960	00:38
1	0.035538	0.030785	0.991284	0.991284	00:38
2	0.020503	0.022382	0.994189	0.994189	00:38
3	0.018317	0.019581	0.994770	0.994770	00:38
4	0.013616	0.020740	0.994189	0.994189	00:38

Fig 24: Evaluation metrics for LSTM

Bi-Directional Encoder Representations from Transformers (BERT)

BERT is a transfer learning model built by the Google AI team in 2018. Before BERT, the common method of training the model was using the feed-forward method which was unidirectional. An improvement to this technique was to use two pipelines, one that moved forward and one that moved backward through the layers so that the model would learn the left and the right contexts of each word in the sentence. But this method did not look both ways at the same training epoch. BERT essentially solved this fundamental problem by learning the position of each word using a positional embedding unique to each word thus understanding the intrinsic meaning of a particular word in relation to its sentence.

BERT incorporates other NLP methodologies such as the architecture of Transformers which uses the Attention mechanism that allows the model to place more weightage on critical words of a sentence to predict the next one. It essentially breaks up a sentence into bite-sized manageable parts, then translates it into its Language Model, then concatenates the smaller parts together to try and understand the whole sentence. Moreover, since BERT looks at the whole sentence instead of trying to read the sentence left-to-right in order to predict the next word, BERT uses a special "MASK" token to cover up words randomly in the sentence and then try to predict those words which leads to better contextual understanding of the language.

In our project, we used the BERT-Base model which has 110 million train parameters. Since it had already been trained on a large corpus of documents from the internet, we found it much easier to fine-tune it on our smaller dataset of political tweets. We found this to give us the best accuracies in the test dataset out of all the above models we tried and hence this is what we used as our final model.

```
F1 score:  0.7448979591836734
Accuracy:  0.7373737373737373
Precision:  0.7525773195876289
Recall:  0.7373737373737373
```

Fig 25: Evaluation metrics for BERT

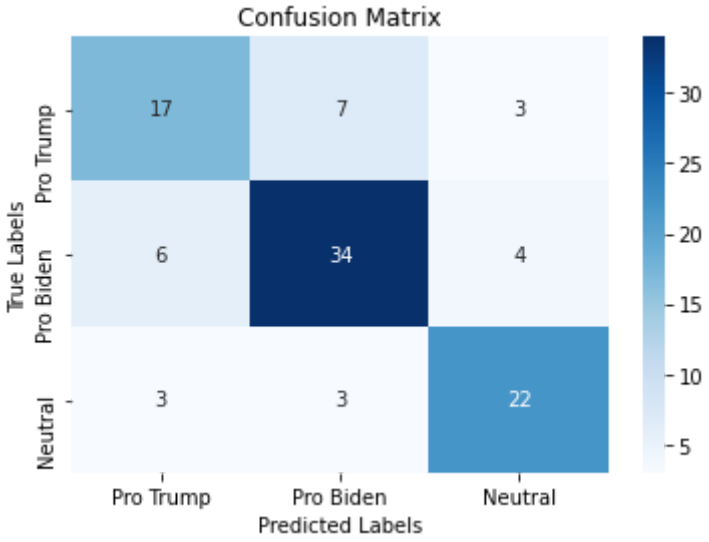

Fig 26: Confusion matrix for BERT

4.2 Discuss whether you had to preprocess data and why

After crawling Twitter, a large corpus of tweets was initially obtained. However, to increase the accuracy of sentiment analysis on the tweets, date pre-processing was required to remove irrelevant tweets and obtain a more meaningful (albeit slightly smaller) corpus. For our project, particularly for the Deep Learning models, data pre-processing involved four steps: Data Cleaning, Tokenization, Numericalization, and Creation of an Embedding Matrix. These steps are detailed in the following sections.

4.2.1 Data Cleaning

As mentioned in section 2.3 of this report (for Crawling), the initial corpus had more than 100,000 tweets after iterating over the keywords/hashtags that were relevant to the US election. However, roughly half the tweets (from observational experience) were either unsatisfactory for the purpose of classifying tweets as either Pro-Trump or Pro-Biden. These tweets were usually completely unrelated to the elections, but contained lots of hashtags to broaden the reach of the tweet (promotional tweets or scam tweets, for example). There were also several duplicate tweets and multiple tweets from the same user.

To clean this data, we removed all tweets which had more hashtags than normal words (words without the hashtag prefix), and removed duplicate tweets based on the content of the tweets. We also kept only one tweet per user to ensure some very-active twitter users do not give an inaccurate representation of sentiment of the country.

Furthermore, the original data contained several columns which we did not consider relevant to the Information Retrieval system or for sentiment analysis, such as “Mentioned Users”, “Out Links”, “Conversation ID”, “Source Label”, etc. These columns were removed during data cleaning, too.

After data cleaning, we were left with a dataset of 38,364 tweets/records, and 8 columns: *username, content, date, country, replyCount, likeCount, retweetCount, and url*.

4.2.2 Tokenization

Tokenization refers to the process of converting the documents into a single list of words. An external python library called *sentencepiece* was used to do the tokenization. It enables word-based tokenization which splits the sentence on a space while also following general language rules like changing [don't] to [do][n't]. It also adds special tokens such as the following:

- xxbos: Indicates the beginning of a text
- xxmaj: Indicates the next word begins with a capital
- xxunk: Indicates the word is unknown (unavailable in the vocabulary of the language model)

It also follows other rules such as:

- replace_rep: Replaces any character repeated three times or more with a special token for repetition (xxrep)
- rm_useless_spaces: Removes all repetitions of the space character
- replace_all_caps: Lowercases a word written in all caps and adds a special token for all caps (xxup) in front of it
- replace_maj: Lowercases a capitalized word and adds a special token for capitalized (xxmaj) in front of it

```
# Tokenize all the rows
tok = Tokenizer.from_df(train)
tok.setup(train)

toks = txts.map(tok)
toks[0]
```

```
(#17) ['xxbos','xxmaj','on','xxmaj','trump',"'s",'superiority','theory',':','\n'...]
```

Fig 27: Code snippet of Tokenization

4.2.3 Numericalization

It is the process of making a list of unique words in the corpus and assigning a number to it based on its index in the vocabulary. In other words, numericalization refers to mapping the tokens from the previous step to an integer. First, a list of all the unique tokens in the corpus

is made. Then, each token is replaced by its index in the vocabulary. First all the special tokens will appear followed by every unique word in the text in a descending frequency order. *Sentencepiece* constricts the maximum number of words in the vocabulary to 60,000 to avoid the embedding matrix from getting too large. So any word that crosses this threshold is stored as an unknown (xxunk) token.

```
# Numericalize all the tokens from the previous step
num = Numericalize()
num.setup(toks)
nums = toks.map(num)
nums[0][:10]
```

```
TensorText([ 2,  8, 44,  8, 18, 47,  0,  0, 13, 27])
```

Fig 28: Code snippet of Numericalization

4.2.4 Creation of an Embedding Matrix

An Embedding Matrix consists of two arrays. The first array contains the independent variable which is the sequence of words starting with the first word in our corpus and ending with the second to last word in the corpus, and the dependent variable will be the sequence of words starting with the second word and ending with the last word. The second array contains the next word to be predicted by the language model and is essentially the first one offset by one to the right.

This step is done automatically by the dataloader of the model.

```
# Create an iterator of our data with torch DataLoader. This helps save on memory during training because, unlike a for loop,
# with an iterator the entire dataset does not need to be loaded into memory

train_data = TensorDataset(train_inputs, train_masks, train_labels, train_token_types)

train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=batch_size)

validation_data = TensorDataset(validation_inputs, validation_masks, validation_labels, validation_token_types)
validation_sampler = SequentialSampler(validation_data)
validation_dataloader = DataLoader(validation_data, sampler=validation_sampler, batch_size=batch_size)
```

Fig 29: Code snippet of Embedding Matrix

4.3 Train Dataset

A total of around 1800 tweets were used to train the model. We initially tried using a balanced dataset with a 1-1-1 ratio and and 1-1-2 ratio for Pro-Biden to Pro-Trump to Neutral Ratios. Additionally, we balanced the data for each country evenly by collecting as close to 40-40-40 tweets/country for the 1-1-1 split and 40-40-80 tweets/country for the 1-1-2 split.

The dataset that gave the best performance was, surprisingly, the unbalanced model with a 1-1.3-1.8 split. This suggests that our model performance is still limited by the number of tweets, and increasing the training data size should significantly improve our model performance. In fact, just by switching from the balanced to unbalanced datasets, we saw around a 5-7% improvement in model performance.

Despite this obvious finding that our model was capped by the number of tweets, we decided not to scrape more since labelling tweets was very time-consuming and required significant efforts from all of us, and we intend to improve the model in our future exploration on this topic.

We also explored multiple train-validation-test splits. The ones that performed best with BERT were the 80-10-10 and 80-15-5. The model that gave the best performance had a split of 80-15-5.

4.4 Post-Classification Dataset

Our model predicts classes of “Pro Biden”, “Pro Trump” or “Neutral”. For our project, we look at how a country inclines towards Biden or Trump, so the neutral tweets are not important and are discarded. This data after removal is indexed and used by our Information Retrieval system for searching and querying.

We also incorporated a weighted sentiment feature in order to increase the weightage of more influential tweets from each country. This is discussed in more detail in the innovation section of this report (Section 5.1).

4.5 Build an evaluation dataset by manually labeling 10% of the collected data

Inter-annotator agreement is a measure of how well two (or more) annotators can make the same annotation decision for a certain category. We use two IAA metrics for qualitative annotations: Cohen and Fleiss' kappa. Cohen kappa is calculated between a pair of annotators and Fleiss' kappa over a group of multiple annotators.

As required, the 6 members of our group are the 6 judges who performed the labelling. In total, we manually labelled 2000 records. The following code snippet shows the calculation of the Kappa score between 2 judges as well as between 6 judges.

```
judge_1 = labelled_data['judge_1'].to_numpy()
judge_2 = labelled_data['judge_2'].to_numpy()

from sklearn.metrics import cohen_kappa_score
cohen_kappa = cohen_kappa_score(judge_1,judge_2)
print("Cohen's kappa score b/w 2 raters : {}".format(cohen_kappa))
```

```
Cohen's kappa score b/w 2 raters : 0.9368047348358289
```

Fig 30: Cohen's Kappa Score Calculation

```
from nltk import agreement

judge_1 = labelled_data['judge_1'].to_numpy()
judge_2 = labelled_data['judge_2'].to_numpy()
judge_3 = labelled_data['judge_3'].to_numpy()
judge_4 = labelled_data['judge_4'].to_numpy()
judge_5 = labelled_data['judge_5'].to_numpy()
judge_6 = labelled_data['judge_6'].to_numpy()

n_annotators =[[0,str(i),str(judge_1[i])]
               for i in range(0,len(judge_1))] + [[1,str(i),str(judge_2[i])]
               for i in range(0,len(judge_2))] + [[2,str(i),str(judge_3[i])]
               for i in range(0,len(judge_3))] + [[3,str(i),str(judge_4[i])]
               for i in range(0,len(judge_4))] + [[4,str(i),str(judge_5[i])]
               for i in range(0,len(judge_5))] + [[5,str(i),str(judge_6[i])]
               for i in range(0,len(judge_6))]
final_rating =  agreement.AnnotationTask(data = n_annotators)
print("Fleiss Kappa score b/w 6 raters is:"+ str(final_rating.multi_kapp()))
```

```
Fleiss Kappa score b/w 6 raters is: 0.9355316248723237
```

Fig 31: Fleiss's Kappa Score Calculation

Since the Kappa Measure is approximately 0.9370, which is above 0.8(80%), our evaluation dataset is considered as a good inter-annotator agreement.

4.6 Provide evaluation metrics

Here is an overall summary of the performance of the models we experimented.

Model Name	F1 Score	Accuracy	Precision	Recall
Gaussian Naive Bayes	0.52808	0.41414	0.59494	0.47475
Random Forest	0.58974	0.46465	0.80702	0.46465
Extra Trees Classifier	0.59036	0.49495	0.73133	0.49495
LSTM	0.56300	0.56300	0.68921	0.46454
BERT	0.74490	0.73737	0.75258	0.73737

4.7 Discuss performance metrics

4.7.1 Average Tweet Length

The average length of a tweet is 21.3 words.

4.7.2 Average Record Classification Metrics

We look at the time taken for running the BERT model on the tweets, and break down the time taken to tokenize the tweets and time to make predictions on the tweets. These operations were performed on the 28,413/38364 records.

Operation	Time to Run Code	Records/sec
Tokenization of Tweets	24s	1,600
Prediction of Tweet Class	397.2s	96.6

4.8 Discuss performance metrics

4.8.1 Tweet Distribution by Class

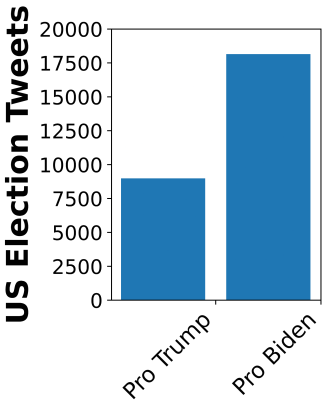

Fig 32: Distribution of classes

The number of Pro Trump tweets, overall, seem to be about half the Pro-Biden tweets. As a whole, the world seems to prefer Biden over Trump. It should be noted that Twitter likely has a Biden bias as it primarily consists of younger users who are traditionally more liberal.

4.8.2 General Word Cloud

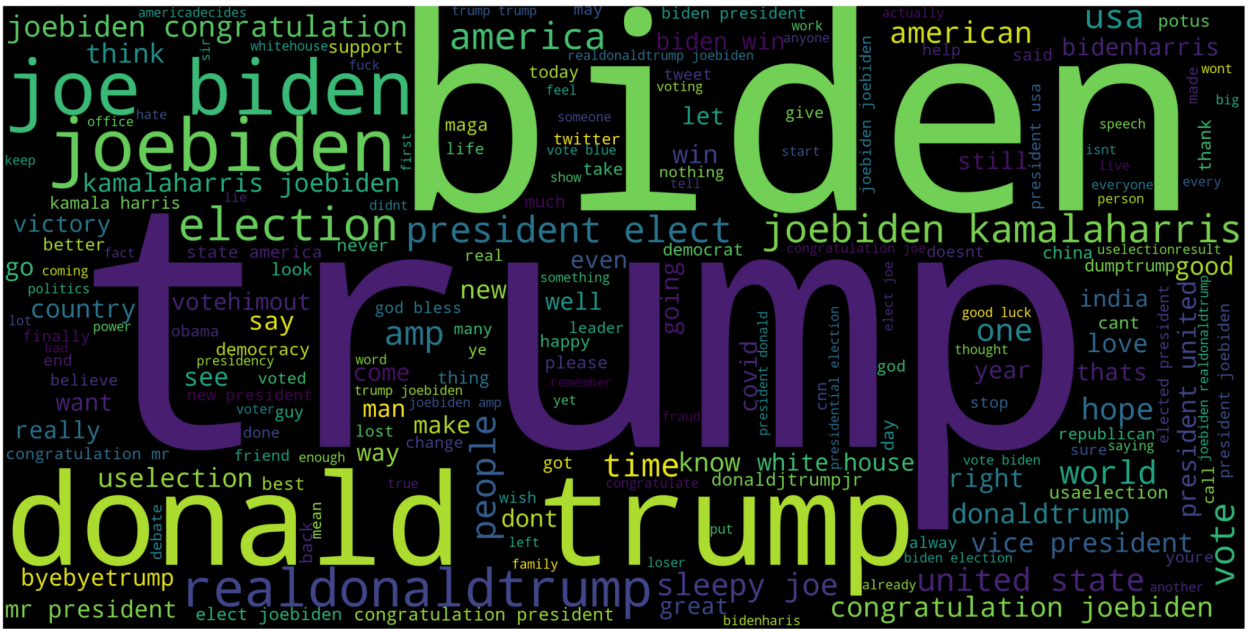

Fig 33: General word cloud

4.8.3 Pro-Trump Word Cloud

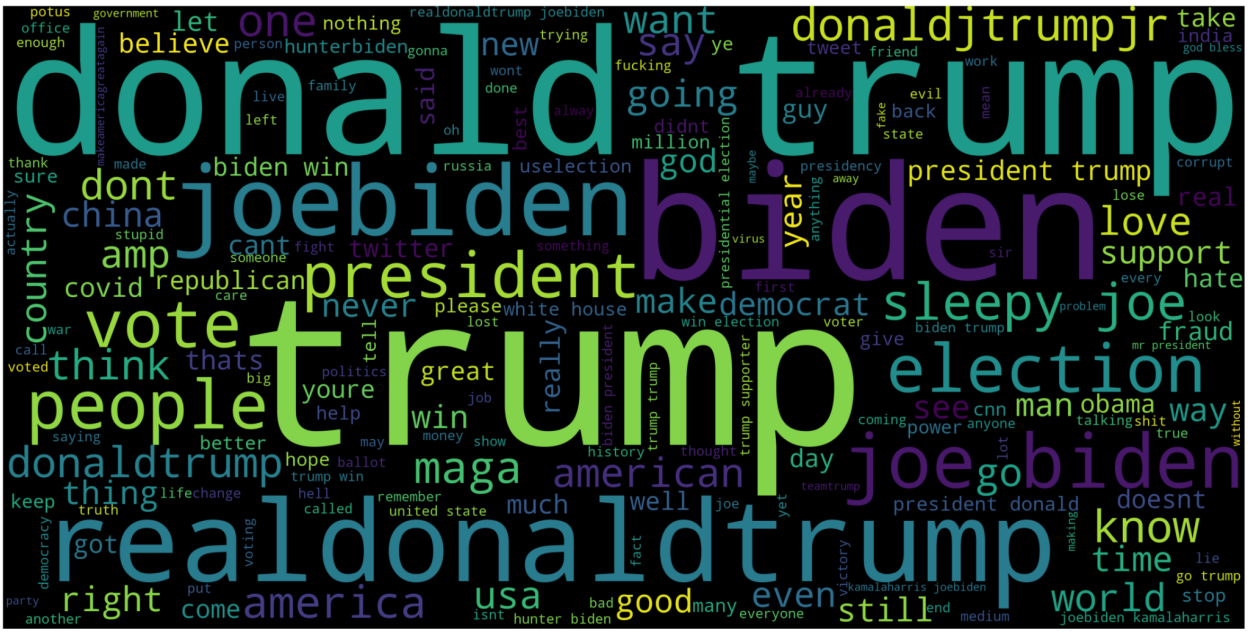

Fig 34: Pro-trump word cloud

4.8.4 Pro-Biden Word Cloud

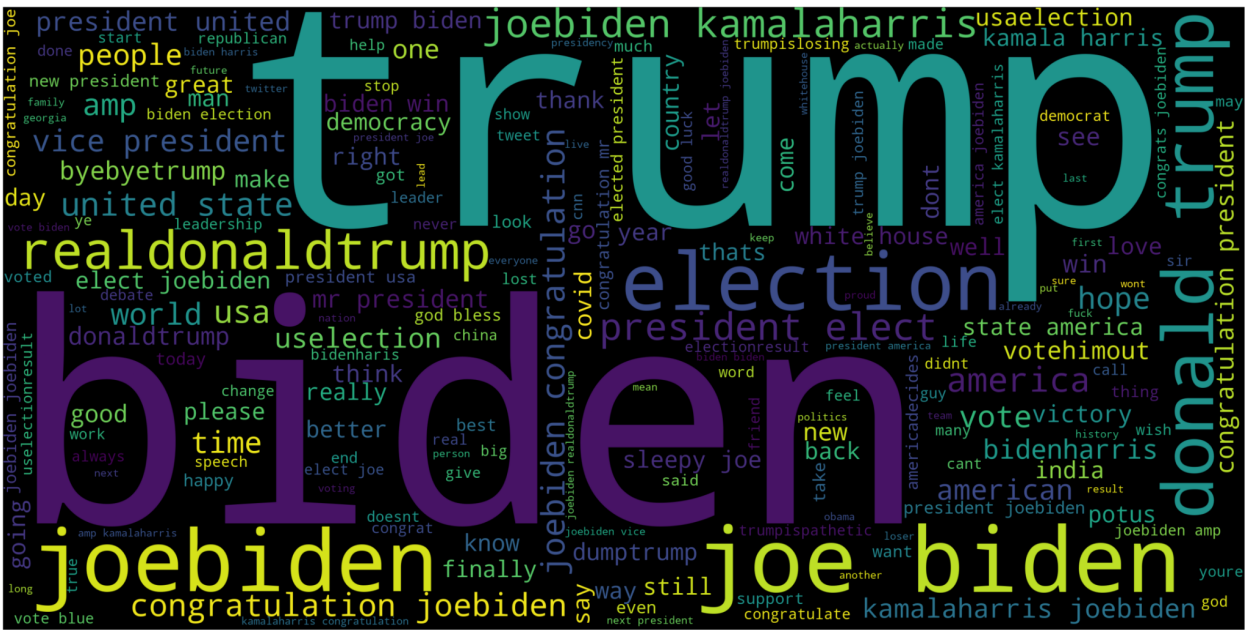

Fig 35: Pro-biden word cloud

4.8.5 Latent Dirichlet Allocation

The Latent Dirichlet Allocation (LDA) graphs help identify the major topics in the data. We use this to gain interesting insights into how the data is clustered. A total of 10 topics were chosen, with the 30 most common terms in each of the topics. The lambda value is used to identify the relevance on terms. The lower the lambda, the more 'rarer' terms belonging to only this cluster are given preference too. This is similar to the tf and idf rankings. With a lower lambda, idf is more important; whereas with a higher lambda is tf is more important.

4.8.5.1 Topic 1

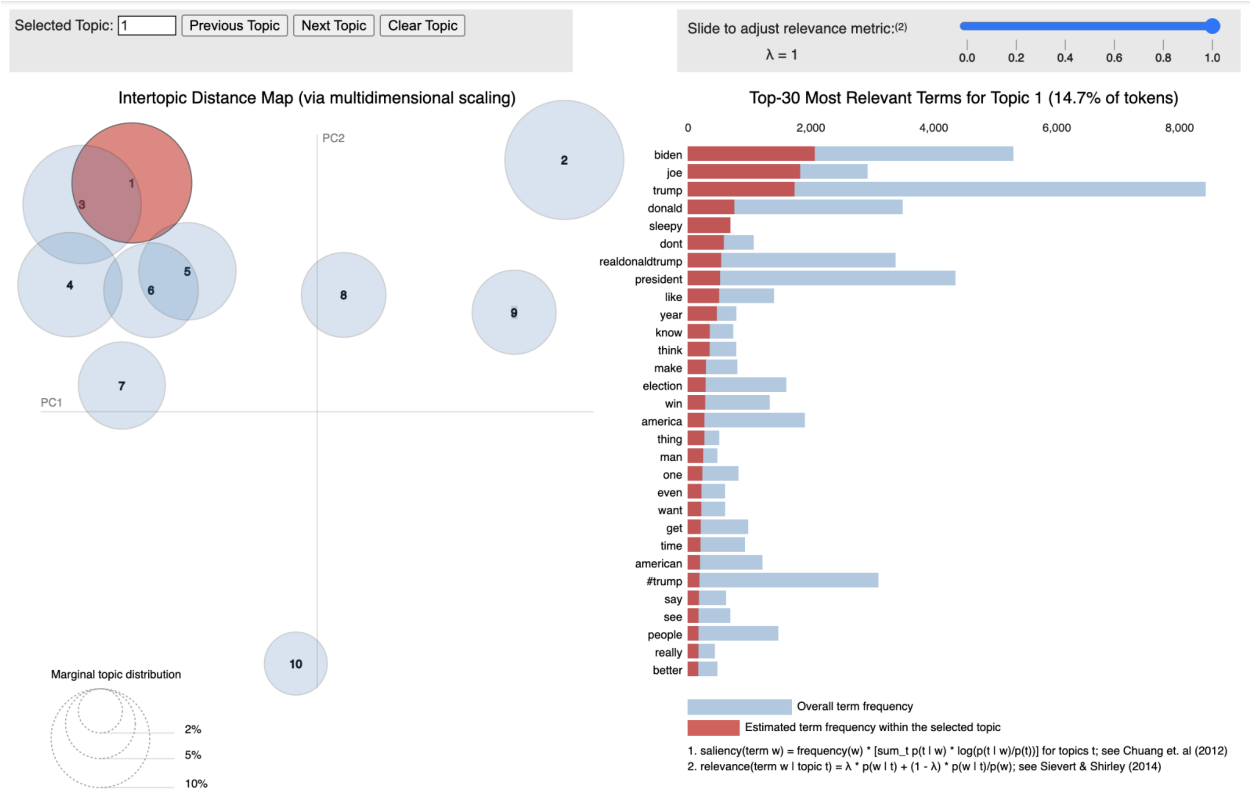

Fig 36: LDA Topic 1

In this first graph, we look at the largest topic. When we look at only the tf in topics (since lambda is 1), we see that the topic contains a good mix of both Trump, Biden and other generic terms and it is hard to understand what the graph is about.

However, when we reduce lambda, we get a clearer insight in the graph below that this topic has a lot of anti-biden tweets with terms such as ‘Sleepy’, which refers to the common insult towards Biden, and ‘Hunter’ which pertains to a scandal against the Biden Family.

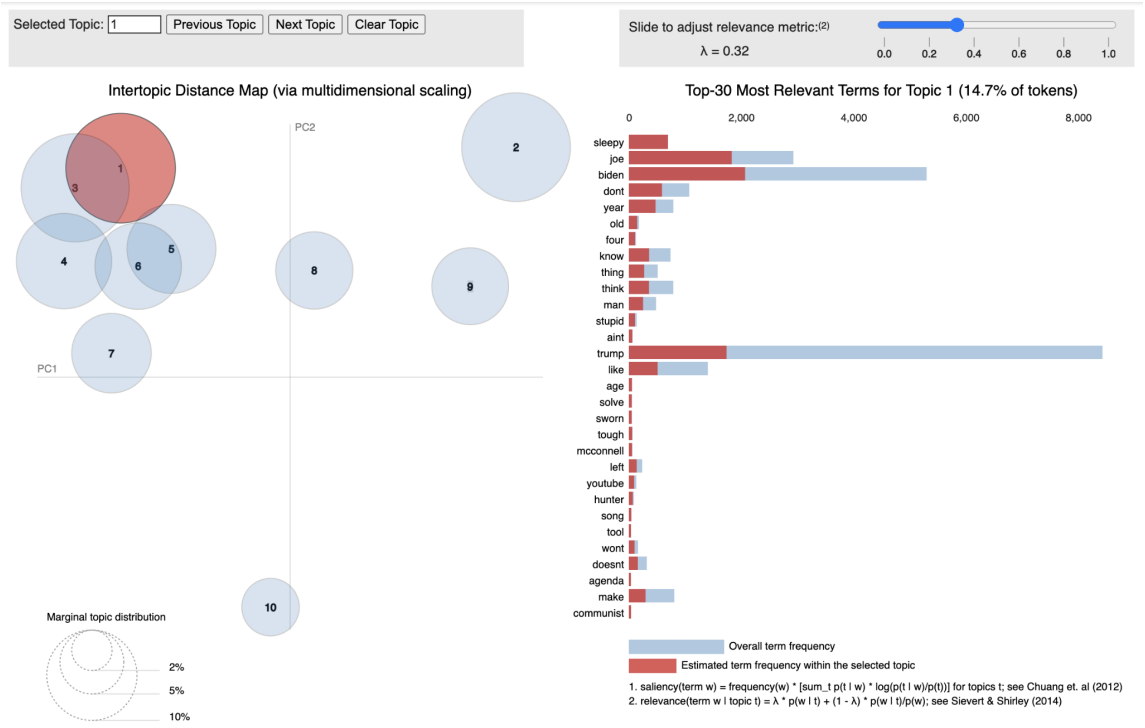

Fig 37: LDA Topic 1

4.8.5.2 Topic 2

The second topic is very far away from the first topic. This is an indication that the tweets in this topic are semantically different from Topic 1. This is indeed the case when we look at the breakdown. The tweets are overwhelmingly positive towards Biden, with terms like “congratulations”, “luck” and hashtags such as “#joebidenkamalaharris”.

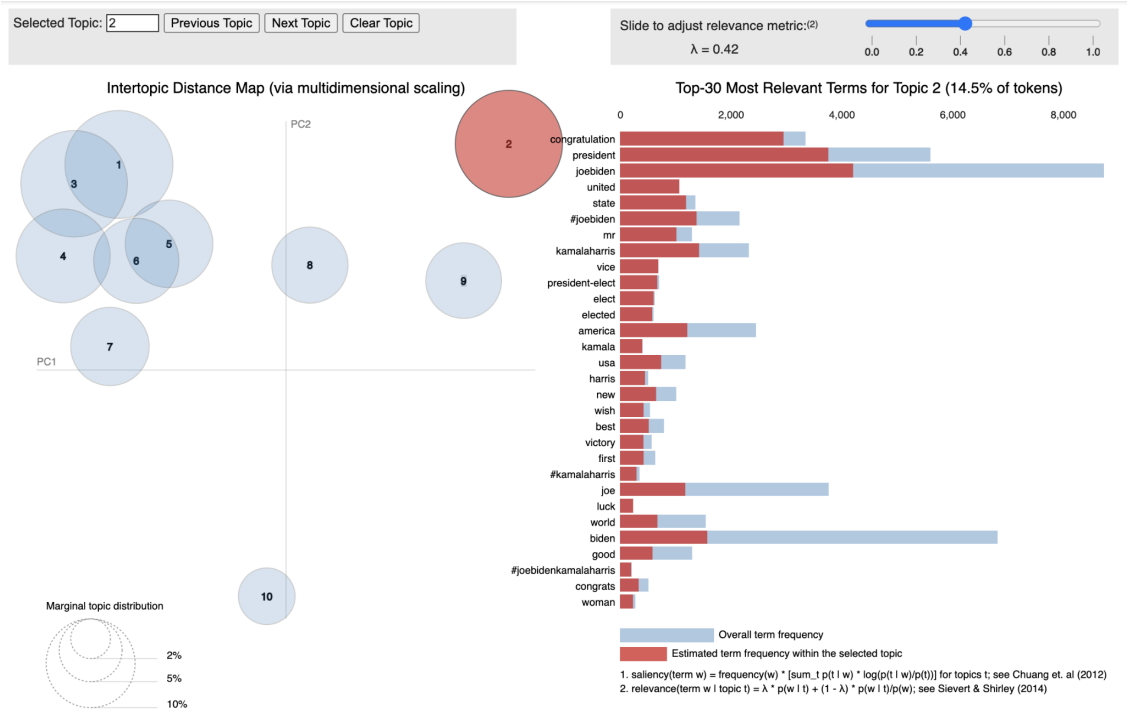

Fig 38: LDA Topic 2

4.8.5.3 Topic 3

This topic is similar to Topic 1 semantically since it is close to Topic 3. What is interesting is that these tweets claim for election fraud with tweets such as “court”, “evidence” and “fraud”, which suggests the tweets are against the Biden victory. While they are similar in intent, they are not the same as the Topic 1 tweets, as topic 1 was explicitly more anti-Biden in nature.

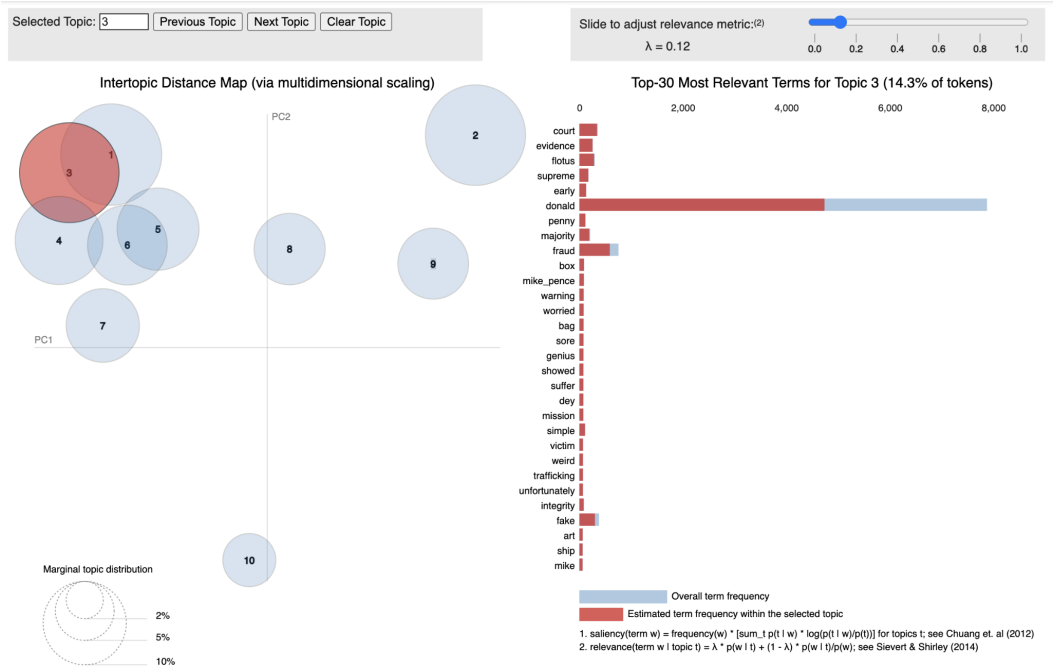

Fig 39: LDA Topic 3

4.8.5.4 Topic 4

This topic seems to contain topics that are a hot region of debate for both parties. Tweets such as “masks”, “virus” and “hoax” appearing at the same time with “#presidentialdebate” and “#trumpisalaughingstock” lend credence to this theory.

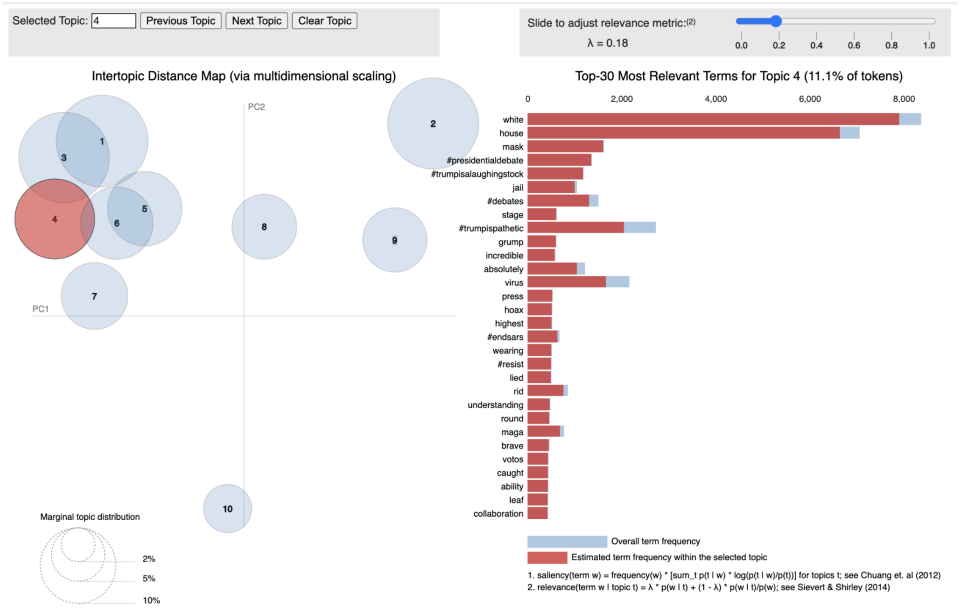

Fig 40: LDA Topic 4

4.8.5.5 Topic 5-10

We could not gain many meaningful insights from the other topics, but they appeared to have a good mix of both Pro-Trump and pro-Biden terms and hashtags.

4.9 Classification Distributions

The table below shows the breakdown of classification grouped by country and sentiment. The ‘count’ column displays the number of tweets per classification. The ‘sum’ column displays the weighted sentiment of the classification. The details to calculate weighted sentiment are given in the next section.

Because of the weighted sentiment scores, we observed some interesting results. Although China (Hong Kong), Ukraine, and Japan have fewer Pro-Trump tweets than Pro-Biden tweets, the overall Pro-Trump weighted score is higher. The only Pro-Trump country with more Pro-Trump tweets than Pro-Biden tweets is Saudi Arabia.

The Pro-Biden Countries are Afghanistan, Australia, Brazil, Canada, France, Germany, India, Indonesia, Iran, Israel, Kenya, Mexico, Philippines, Russia, Singapore, Taiwan and UK. All of these countries have significantly more Pro-Biden tweets than Pro-Trump tweets.

country	sentiment	count	sum
Afghanistan	Pro Biden	728	3.060130
	Pro Trump	181	0.331133
Australia	Pro Biden	1059	5.378802
	Pro Trump	731	2.036534
Brazil	Pro Biden	1298	4.373934
	Pro Trump	831	2.130744
Canada	Pro Biden	1151	5.076177
	Pro Trump	767	1.851881
China	Pro Biden	504	1.145054
	Pro Trump	314	1.297892
France	Pro Biden	2328	10.222376
	Pro Trump	1049	2.708367
Germany	Pro Biden	390	1.896711
	Pro Trump	359	1.145072
India	Pro Biden	1699	6.593219
	Pro Trump	623	1.760649
Indonesia	Pro Biden	1038	2.618412
	Pro Trump	252	0.387142
Iran	Pro Biden	220	0.496881
	Pro Trump	140	0.347130
Israel	Pro Biden	636	2.568697
	Pro Trump	220	0.771813
Japan	Pro Biden	224	0.576637
	Pro Trump	113	0.585576
Kenya	Pro Biden	717	3.040431
	Pro Trump	313	1.449003
Mexico	Pro Biden	1393	6.529880
	Pro Trump	444	1.843050
Nigeria	Pro Biden	1056	0.689264
	Pro Trump	692	3.358437
Philippines	Pro Biden	64	0.125730
	Pro Trump	10	0.015117
Russia	Pro Biden	641	2.093741
	Pro Trump	416	1.241686
Saudi Arabia	Pro Biden	142	0.076577
	Pro Trump	185	0.283402
Singapore	Pro Biden	319	1.503000
	Pro Trump	105	0.276576
South Korea	Pro Biden	486	0.905216
	Pro Trump	349	0.924605
Taiwan	Pro Biden	140	0.230932
	Pro Trump	37	0.053982
UK	Pro Biden	1652	6.042433
	Pro Trump	673	2.030618
Ukraine	Pro Biden	269	0.677080
	Pro Trump	188	0.718820

Fig 41: Class distributions after weighted sentiment analysis

5 Explore some innovations for enhancing classification

5.1 Weighted Sentiment after Classification

The utilized classification outputs are of the format “Pro Trump” or “Pro Biden”. We decided the count of the classes alone would not be a great metric for determining if a country is leaning Pro-Trump or Pro-Biden, and calculated a Weighted Sentiment component for comparisions. The rationale behind this was that all tweets are not equal. The “likeCount”, “retweetCount” and “replyCount” are an important metric in determining the influence and sentiment of the tweet. A Pro-Trump tweet with more engagement should influence the sentiment more than its Pro-Biden counterpart with fewer likes.

```
def logFinder(df):
    if df == 0:
        x = 1
    else:
        x = 1+np.log(2*df)
    return x
```

```
rep = df['replyCount'].apply(lambda x: logFinder(x))
ret = df['retweetCount'].apply(lambda x: logFinder(x))
lik = df['likeCount'].apply(lambda x: logFinder(x))

df['weightedSentiment'] = 1*rep*ret*lik
```

```
a, b = 0, 1
x, y = df.weightedSentiment.min(), df.weightedSentiment.max()
df['weightedSentimentNorm'] = (df.weightedSentiment - x) / (y - x) * (b - a) + a
```

Fig 42: Code snippet for weighted sentiment analysis

The weighting, in the code above, is inspired by the tf-idf formula. We take the logarithm of 2 x likes, 2 x retweets, 2 x replies. The multiple of 2 is to differentiate between 0 and 1 likes, as in the logFinder function, there are no changes made if likes, retweets or replies are 0. However, log(1) = 0, so which means without the multiple of 2, no differentiation is made between 1 like and 0 likes.

These three logarithms of attributes (likes, replies, and retweets) are multiplied and stored in the ‘weightedSentiment’ column. Finally, this column is normalized between 0 and 1. By

grouping the tweets by country and sentiment (Pro Trump or Pro Biden) and taking a sum of Weighted Sentiment, we get a two Weighted Sentiment scores for each country, one Pro-Trump and one Pro Biden. These scores represent how “Pro Trump” or “Pro Biden” a country is.

The weighted sentiment score for each country is taken for both classes and the score of Pro Trump is normalized to between 0 and 0.5 (closer to 0 implies more Pro Trump) and the score of Pro Biden is normalized between 0.5 and 1. The dominating value among the both is taken and is visualized in a world choropleth map. This map is used to determine which country is leaning Pro Trump or Pro Biden, and the percentage is also displayed when you hover over a particular country. In this map, the color red represents Pro Trump and color blue represent Pro Biden. This map is created using plotly and pycountry in Python.

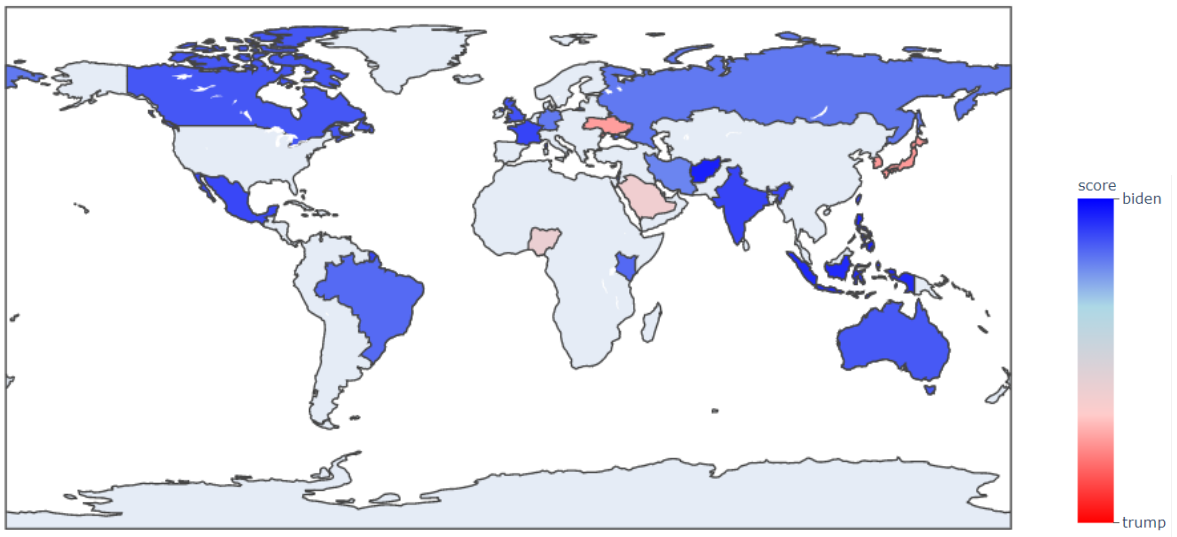

Fig 43: Weighted Sentiment Distribution of ‘Pro Trump’ or ‘Pro Biden’ on the World Map

5.2 Optimizing threshold value for better f1-score

The BERT model gives a prediction as a probabilistic value which ranges from 0 to 1 which represents its confidence in its prediction. In our model, we had 3 labels (Pro Trump, Pro Biden, Neutral) and that resulted in 3 confidence values, one for each class. These confidence values were further converted to prediction booleans for measuring performance. This is done by setting a threshold value for the confidence and if the confidence is higher than the threshold value, then the prediction is converted to a positive boolean value. The positive

boolean value represents that the tweet belongs to that particular class. This threshold value is used as a variable ranging from 0.1 to 0.9 at steps of 0.05 in the validation dataset, to find the optimal threshold value for better f1-score.

```
# Calculate Accuracy - maximize F1 accuracy by tuning threshold values. First with 'macro_thresholds' on the order of e^-1 then with 'micro_thresholds'

macro_thresholds = np.array(range(1,10))/10

f1_results, flat_acc_results = [], []
for th in macro_thresholds:
  pred_bools = [pl>th for pl in pred_labels]
  test_f1_accuracy = f1_score(true_bools,pred_bools,average='micro')
  test_flat_accuracy = accuracy_score(true_bools, pred_bools)
  f1_results.append(test_f1_accuracy)
  flat_acc_results.append(test_flat_accuracy)

best_macro_th = macro_thresholds[np.argmax(f1_results)] #best macro threshold value

micro_thresholds = (np.array(range(10))/100)+best_macro_th #calculating micro threshold values

f1_results, flat_acc_results = [], []
for th in micro_thresholds:
  pred_bools = [pl>th for pl in pred_labels]
  test_f1_accuracy = f1_score(true_bools,pred_bools,average='micro')
  test_flat_accuracy = accuracy_score(true_bools, pred_bools)
  f1_results.append(test_f1_accuracy)
  flat_acc_results.append(test_flat_accuracy)

best_f1_idx = np.argmax(f1_results) #best threshold value

# Printing and saving classification report
print('Best Threshold: ', micro_thresholds[best_f1_idx])
print('Test F1 Accuracy: ', f1_results[best_f1_idx])
print('Test Flat Accuracy: ', flat_acc_results[best_f1_idx], '\n')
```

Fig 44: Code snippet for optimizing threshold value

5.3 Data Augmentation

We were facing an overfitting issue while trying out the LSTM model. In order to try and fix that, we decided to artificially increase the size of our train dataset by performing data augmentation. We used a library called Easy Data Augmentation (EDA) which allowed us to create 10 different augmentations for each tweet. These were the available augmentations:

- Synonym Replacement (alpha_sr): Randomly replace n words from the sentence that are not stop words with one of its synonyms
- Random Insertion (alpha_ri): Insert a synonym of a random word in the sentence that is not a stop word into a random position in the sentence
- Random Swap (alpha_rs): Randomly choose 2 words in the sentence and swap them
- Random Deletion (alpha_rd): Randomly remove a word in the text

Of the above augmentations, we chose to do Synonym Replacement and Random Insertion because those 2 did not change the overall meaning of the sentence, it only made it wordier in length. Since the tweets were already a few characters in length, we decided not to do Random Deletion which would shorten the tweet further. Random Swap sometimes would swap critical words in a tweet like proper nouns like ‘Donald Trump’ or ‘Joe Biden’ which was not a good practice while training the model.

```
# Convert csv to txt file required for EDA to run
eda.to_csv('train2.txt', header=False, index=False, sep='\t', mode='a')
```

```
# Run the python command for EDA with 10 new augmentations per existing tweet and perform all 4 augmentations with its default al
%run eda_nlp/code/augment.py --input=eda_nlp/data/train2.txt --num_aug=10 --alpha_sr=0.05 --alpha_rd=0.00 --alpha_ri=0.05 --alpha
```

```
generated augmented sentences with eda for eda_nlp/data/train2.txt to eda_nlp/data/eda_train2.txt with num_aug=10
```

```
# Convert txt to csv and replace train dataset with new EDA data
train = pd.read_csv('eda_train2.txt', delimiter="\t", header=None, names=["classes", "content"])
```

Fig 45: Code snippet for data augmentation

6 Conclusion

In our project, our group has crawled through Twitter to build a corpus that is used to gain insights on the preferences of different countries for the 2 candidates of the 2020 US Election. Using SNScrape, the data was crawled for the time period of the 2020 Election. We then created a web-interface using HTML, CSS & hosted using Flask. The website is integrated with the indexing software, Solr, where the required data manipulation, indexing & selection are performed. After which, we proceed with data pre-processing using techniques such as Tokenization and Numericalization. Functionalities such as inputting search queries, sorting the tweets displayed, applying country based filters are performed through an interactive UI implemented using HTML & CSS. Following that, we performed our sentiment prediction using various machine learning classification models such as Naïve Bayes classifier, Random Forests and Extra Trees Classifier. We further improved our sentiment analysis through a Deep Learning approach using LSTM & BERT models. The models are then evaluated using evaluation metrics such as F-measure, Precision and Recall values and the results are compiled and analyzed. To gain further insights into the data, we used libraries such as pyLDAvis, NLTK & wordcloud to generate meaningful data visualizations. Lastly, we explored some innovations to enhance our classification such the use of Weighted Sentiment after Classification, optimizing threshold value for better f1-score, and data augmentation.

7 Bibliography

Video, Data and Source Codes links

The following table will consist of the URL to the various resources used as well as the Youtube video link and the link to the Google Drive which holds our analysis data, analysis results and source codes.

Name	URL
Corpus Webpage (Twitter)	https://www.twitter.com/
Source Code on GitHub	https://github.com/AtrikDas/Info-Retrieval-Group-Project
Documentation for Crawling (Scraping) Tool - SNScrape	https://github.com/JustAnotherArchivist/snscrape
Documentation for Indexer Used - Solr	https://solr.apache.org/guide/
Documentation for Classification Model - BERT	https://huggingface.co/transformers/model_doc/bert.html
Crawling Source Code & Data (Google Drive Link)	https://drive.google.com/drive/folders/1IK_ffGbR_a47qW7W2eMo55TRw6BcGOK-?usp=sharing
Classification Source Code & Data (Google Drive Link)	https://drive.google.com/drive/folders/1-ZOHW0z9yPXD-OKWe8AJde_skZ29GFo8?usp=sharing
Youtube Link to Our Video Presentation	https://www.youtube.com/watch?v=uETnKOS5VQ4

8 Work Distribution

ROLES	MEMBER
Crawling and Data Preparation	Padhi Abhinandan, Das Atrik
Indexing & Querying	Chandrasekhar Aditya, Rajuravi Vishal Raj, Gunturi Kushal Sai
Classification	Gunturi Kushal Sai, Chandrasekhar Aditya, Das Atrik, Padhi Abhinandan
Web-Interface	Chandrasekhar Aditya, Jose Jeswin, Rajuravi Vishal Raj
Video	Chandrasekhar Aditya, Jose Jeswin
Documentation and Labelling of Train Data	All Team Members