NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

CZ4034 Information Retrieval

Course Assignment Report

Group 11

S/N	Team Member Name	Matriculation Number
1	Trinh Tuan Dung	U1720421K
2	Nguyen Doan Hoang Lam	U1720531E
3	Gabriella Benedicta Christianti	U1720401C
4	Stephanie Audrey Susanto	U1720851C
5	Le Tan Khang	U1720161L
6	Shearman Chua Wei Jie	U1820058D

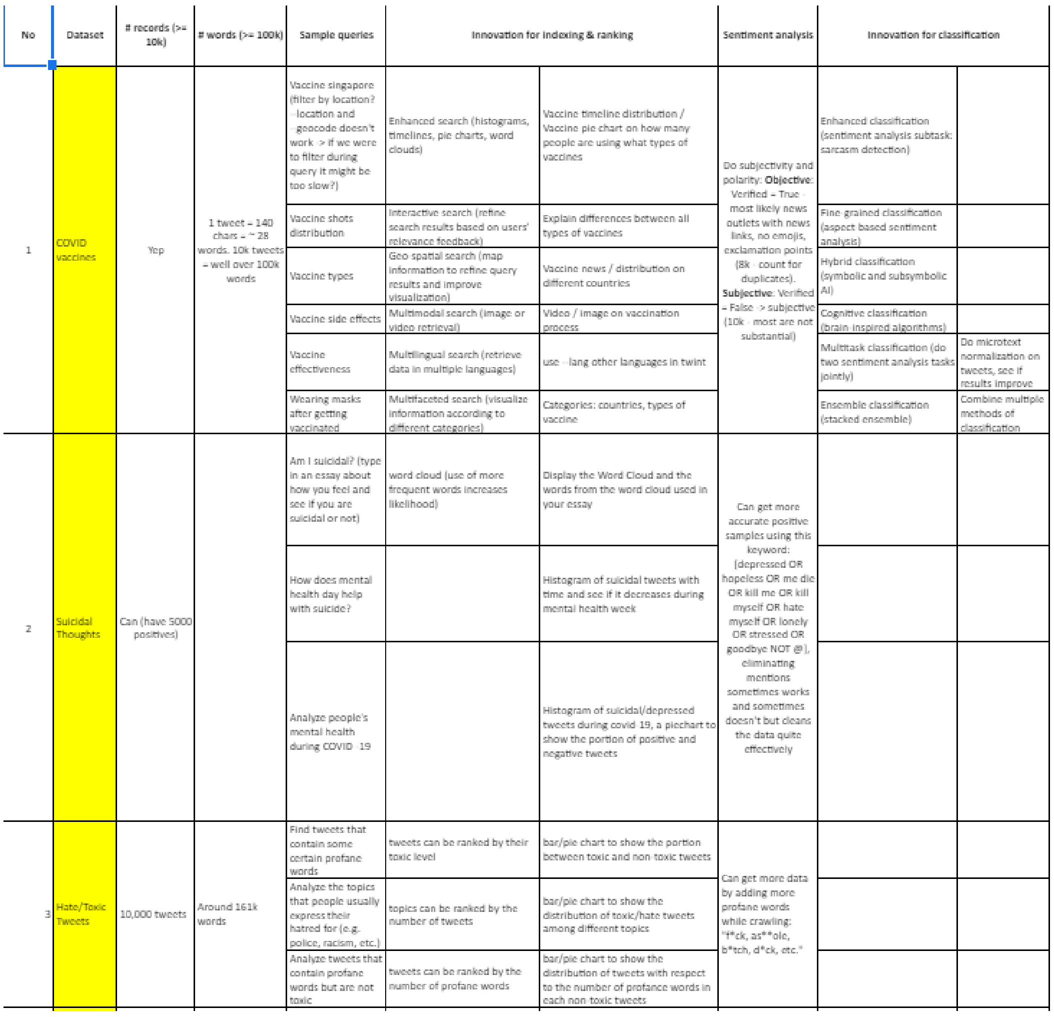
Contents

1. Introduction

For our CZ4034 Information Retrieval Course Assignment, our team is required to build an information retrieval system for sentiment analysis. In particular, we will have to crawl a text corpus based on a topic of our interest, build a search engine to query over the corpus, and finally perform sentiment analysis over it. The system can be about any domain and data of our preference. For example, an information retrieval system for social media marketing, political forecasting, healthcare, financial forecasting, or a recommendation system.

1.1 Selection of Topic for Course Assignment

Before we can proceed to build our information retrieval system, we will need to select a topic of interest for us to build our system on. Our team decided to individually conduct a simple Exploratory Data Analysis on a unique topic each, and with the information gathered from the Exploratory Data Analysis for each of the topic, we will then be able to discuss and decide as a group on which topic to be utilized as the topic of interest for our information retrieval system.



No	Dataset	# records (>= 10k)	# words (>= 100k)	Sample queries	Innovation for indexing & ranking		Sentiment analysis	Innovation for classification	
1	COVID vaccines	Yep	1 tweet – 140 chars – ~ 28 words. 10k tweets – well over 100k words	Vaccine singapore (filter by location? --location and --geocode doesn't work -> if we were to filter during query it might be too slow?)	Enhanced search (histograms, timelines, pie charts, word clouds)	Vaccine timeline distribution / Vaccine pie chart on how many people are using what types of vaccines	Do subjectivity and polarity: **Objective**: Verified – True - most likely news outlets with news links, no emojis, exclamation points (8k - count for duplicates). **Subjective**: Verified – False -> subjective (10k - most are not substantial)	Enhanced classification (sentiment analysis subtask: sarcasm detection)	
				Vaccine shots distribution	Interactive search (refine search results based on users' relevance feedback)	Explain differences between all types of vaccines		Fine-grained classification (aspect based sentiment analysis)	
				Vaccine types	Geo spatial search (map information to refine query results and improve visualization)	Vaccine news / distribution on different countries		Hybrid classification (symbolic and subsymbolic AI)	
				Vaccine side effects	Multimodal search (image or video retrieval)	Video / image on vaccination process		Cognitive classification (brain-inspired algorithms)	
				Vaccine effectiveness	Multilingual search (retrieve data in multiple languages)	use --lang other languages in twint		Multitask classification (do two sentiment analysis tasks jointly)	Do microtext normalization on tweets, see if results improve
				Wearing masks after getting vaccinated	Multifaceted search (visualize information according to different categories)	Categories: countries, types of vaccine		Ensemble classification (stacked ensemble)	Combine multiple methods of classification
2	Suicidal Thoughts	Can (have 5000 positives)		Am I suicidal? (type in an essay about how you feel and see if you are suicidal or not)	word cloud (use of more frequent words increases likelihood)	Display the Word Cloud and the words from the word cloud used in your essay	Can get more accurate positive samples using this keyword: (depressed OR hopeless OR me die OR kill me OR kill myself OR hate myself OR lonely OR stressed OR goodbye NOT @), eliminating mentions sometimes works and sometimes doesn't but cleans the data quite effectively		
				How does mental health day help with suicide?		Histogram of suicidal tweets with time and see if it decreases during mental health week			
				Analyze people's mental health during COVID -19		Histogram of suicidal/depressed tweets during covid-19, a piechart to show the portion of positive and negative tweets			
3	Hate/Toxic Tweets	10,000 tweets	Around 161k words	Find tweets that contain some certain profane words	tweets can be ranked by their toxic level	bar/pie chart to show the portion between toxic and non-toxic tweets	Can get more data by adding more profane words while crawling: "f*ck, as**ole, b*tch, d*ck, etc."		
				Analyze the topics that people usually express their hatred for (e.g. police, racism, etc.)	topics can be ranked by the number of tweets	bar/pie chart to show the distribution of toxic/hate tweets among different topics			
				Analyze tweets that contain profane words but are not toxic	tweets can be ranked by the number of profane words	bar/pie chart to show the distribution of tweets with respect to the number of profance words in each non-toxic tweets			

Figure 1: Selection of Topic CSV file

From the figure, we are able to observe some of the topics that were suggested, as well as the various analysis made on their suitability as a topic to be selected for our information retrieval system. In the end, we as a team selected the topic of hate speech and toxic comments on Twitter as the topic of interest for our information retrieval system.

The first reason for selecting the topic of hate speech and toxic comments on Twitter as the topic of interest for our information retrieval system, is that hate speech and toxic comments can be commonly seen on Twitter which means that we will have sufficient data to be used to build our text corpus. Next, for the topic of hate speech and toxic comments, we will be able to perform multitask classification on the text, classifying their subjectivity as well as toxicity level. Finally, due to the fact that we are required to manually annotate 10 percent of the text corpus for use as validation set for our classifier, it will be laborious to have to manually annotated a large amount of text from our text corpus to be used to train our classification model. For the topic of hate speech and toxic comments, we found a dataset on Kaggle for Toxic Comment Classification, the dataset contains a huge amount of Wikipedia comments which have been annotated by human raters for toxic behavior, therefore, if we were to use the topic of hate speech and toxic comments on Twitter as the topic of interest for our information retrieval system, we will be able to make use of this pre-annotated Kaggle dataset to train our classification model and we can afford to manually annotate less amount of text from the crawled text corpus and still be able to have enough training data to train our classification model. With the reasons stated above, our team selected the topic of hate speech and toxic comments on Twitter as the topic of interest for our information retrieval system.

1.2 Problem Statement

The use of social media platforms such as Twitter has enabled us to be able to freely express our thoughts and ideas online, as well as hyper target, and mass promote content and personal brand around the world. However, the rights of free speech, as well as the ease of posting comments on Twitter gave rise to negative online behaviors, like toxic comments (i.e., comments that are rude, disrespectful, or otherwise likely to make someone leave a discussion). The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.

Therefore, there is a need for tools to detect the presence of negative online behaviors, as well as toxic comments in order to be able to shut down negative online behaviors and toxic comments from social media platforms.

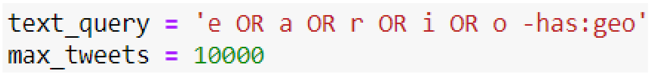
The objective of our team's project is to build an information retrieval system that is able to allow us to query tweets crawled from Twitter, perform sentiment analysis over them using various classification models, and thereafter display the data in a way that will allow us to analyze the presence of toxic comments, their origins as well as their various metadata compositions.

2. Crawling of Text Data

2.1 Methodology for Crawling of Data

Question 1: How you crawled the corpus (e.g., source, keywords, API, library) and stored it?

The data is crawled from Twitter using the Tweepy library which makes use of the Twitter API. Our dataset consists of two main parts. The first part contains 5147 randomly crawled tweets with a Boolean text query containing a single letter. The specific text query for this part is shown in the following picture:

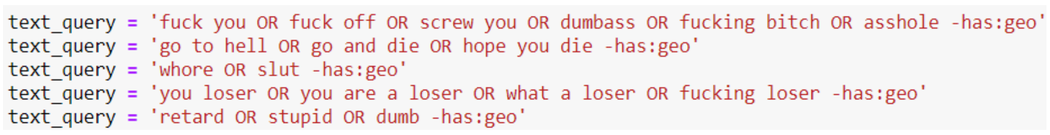


```
text_query = 'e OR a OR r OR i OR o -has:geo'
max_tweets = 10000
```

Figure 2: Crawling Query with Single Letter

To ensure that the tweets crawled for this part are of random nature, the letters chosen in the text query are the five most frequently used alphabet letters in the Concise Oxford English dictionary.

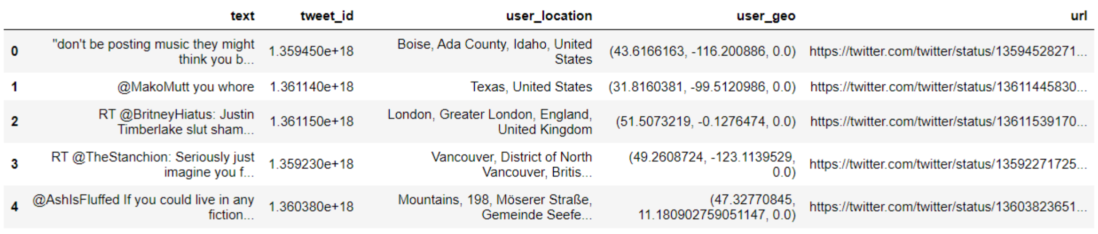
The second part contains 5846 tweets. This part is crawled with the aim of having a large portion being categorized as toxic tweets. Therefore, the Boolean search query for this part contains toxic terms that we defined based on our common sense. This part of the dataset is the combined crawled tweets of five different Boolean text query results. The text queries are shown in the following picture:



```
text_query = 'fuck you OR fuck off OR screw you OR dumbass OR fucking bitch OR asshole -has:geo'
text_query = 'go to hell OR go and die OR hope you die -has:geo'
text_query = 'whore OR slut -has:geo'
text_query = 'you loser OR you are a loser OR what a loser OR fucking loser -has:geo'
text_query = 'retard OR stupid OR dumb -has:geo'
```

Figure 3: Crawling Query with Toxic Related Text

Some samples of our crawled data are shown as follows:



	text	tweet_id	user_location	user_geo	url
0	"don't be posting music they might think you b...	1.359450e+18	Boise, Ada County, Idaho, United States	(43.6166163, -116.200886, 0.0)	https://twitter.com/twitter/status/13594528271...
1	@MakoMutt you whore	1.361140e+18	Texas, United States	(31.8160381, -99.5120986, 0.0)	https://twitter.com/twitter/status/13611445830...
2	RT @BritneyHiatus: Justin Timberlake slut sham...	1.361150e+18	London, Greater London, England, United Kingdom	(51.5073219, -0.1276474, 0.0)	https://twitter.com/twitter/status/13611539170...
3	RT @TheStanchion: Seriously just imagine you f...	1.359230e+18	Vancouver, District of North Vancouver, Britis...	(49.2608724, -123.1139529, 0.0)	https://twitter.com/twitter/status/13592271725...
4	@AshIsFluffed If you could live in any fiction...	1.360380e+18	Mountains, 198, Möserer Straße, Gemeinde Seefe...	(47.32770845, 11.180902759051147, 0.0)	https://twitter.com/twitter/status/13603823651...

Figure 4: Sample Crawled Text Data from Twitter

2.2 Analysis of Crawled Data in Text Corpus

Question 1: The numbers of records, words, and types (i.e., unique words) in the corpus.

Our combined dataset has a total of 10993 tweets crawled from Twitter. The dataset is stored as a CSV file and each record is a tweet corresponding to a line in the csv file. The total number of string tokens in our dataset is 207601. Among those, there are 23686 unique tokens after applying stemming using PorterStemmer from nltk python library. Our crawled data has 13 columns, namely, *tweet_id*, *text, created_date_time, username, user_screen_name, user_id, user_location, user_description, verified, associated_place, retweet_count, user_geo, url*.

Question 1: What kind of information users might like to retrieve from your crawled corpus (i.e., applications), with sample queries

Our crawled text corpus can be useful for users who want to do research or analyze human toxic behavior through their daily tweets. They can also make comparisons and retrieve qualitative results about toxic behaviors with regards to users from different countries or region's cultures. For example, users can query the common toxic term "loser" or "fuck" to survey how popular these terms are used in daily tweets or finding out which region having the greatest number of users possessing toxic behavior.

3. Indexing and Querying of Text Corpus

3.1 Data Indexing and Querying using Apache Solr

In our project, we use Apache Solr to index the crawled data. Solr is an open-source enterprise-search platform, written in Java. Some major features in Solr include full-text search, real-time indexing, tokenization, stop-word removal, stemming, and spell checking. While Solr provides many different APIs (Application Programming Interfaces), its REST (Representational State Transfer) API is the easiest way to send data to Solr. We can communicate with the Solr server using HTTPS methods such as GET, POST, PUT, and DELETE, with payloads formatted in JSON. In this section, we will describe how we index the data and configure useful features, such as spell checking and stemming, in Solr.

3.1.1 Data Fields

After getting the crawled data, we select fields and their content from the raw data required to be added to Solr, which are as follows:

Field	Description	Example content
id	Unique identifier	0
tweet	Text content of the tweet	"RT @TrumpWarRoom___: RT to wish a Happy President's Day to Donald Trump.\r\n\r\nFollow @realGOPsupport"
user_location	Location of user	"Virginia, United States"
link	Link to the tweet	"https://twitter.com/twitter/status/1361390039108907008"
user_geo	Longitude and latitude values of the location	"37.1232245", "-78.4927721", "0.0"
toxicity	Toxicity level of the tweet (0, 1, or 2)	0
subjectivity	Subjectivity level of the tweet (0 or 1)	1

Table 1: Data Fields for Tweet Data

3.1.2 Solr Configurations

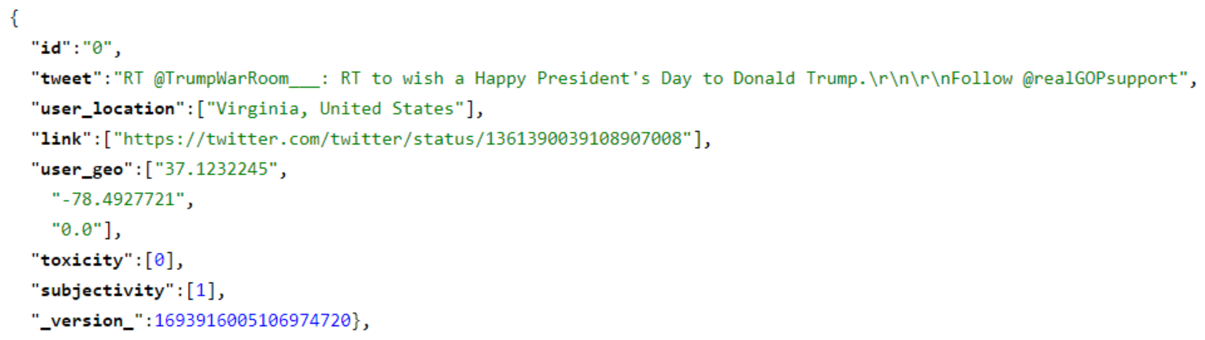
After acquiring the data, we need to perform some extra configuration steps in Solr before indexing. Particularly, Solr does not automatically include the spell checking and stemming features. In order to improve the search experience of our search system for the users, we need to conduct the stemming for both indexing and searching such that the search system can return more relevant documents containing the "root" of each word in the query text. Furthermore, spell checking is also helpful as it provides suggestions for users when their inputs return no matching documents.

With regard to the stemming configuration, we define an additional field type, named "*text_gen_stem*", in the "*managed-schema*" file. In this field type, we also include the fundamental text preprocessing steps, such as tokenization, stop-word removal, and case folding. Eventually, we add the filter "*solr.PorterStemFilterFactory*" to enable stemming while indexing and searching.

On the other hand, to enable spell checking, we first define an additional field, named "*spellcheck*", in the "*managed-schema*" file. Specifically, this field is a duplicate of the "*tweet*" field for Solr to perform the spell checking. Subsequently, we modify the "*solr.SpellCheckComponent*" to use the spell checker named "*solr.IndexBasedSpellChecker*" such that Solr will only generate and suggest terms that are actually present in our search corpus. Finally, we include the "*spellcheck*" field, which will provide search suggestions if users' queries return no result, in the query response.

3.1.3 Data Indexing in Solr

After all the pre-processing and configuration steps, we are now ready to index the data into Solr. We first restart the Solr server and then send all the data through a POST request. An example of the index data in Solr admin is as follows:



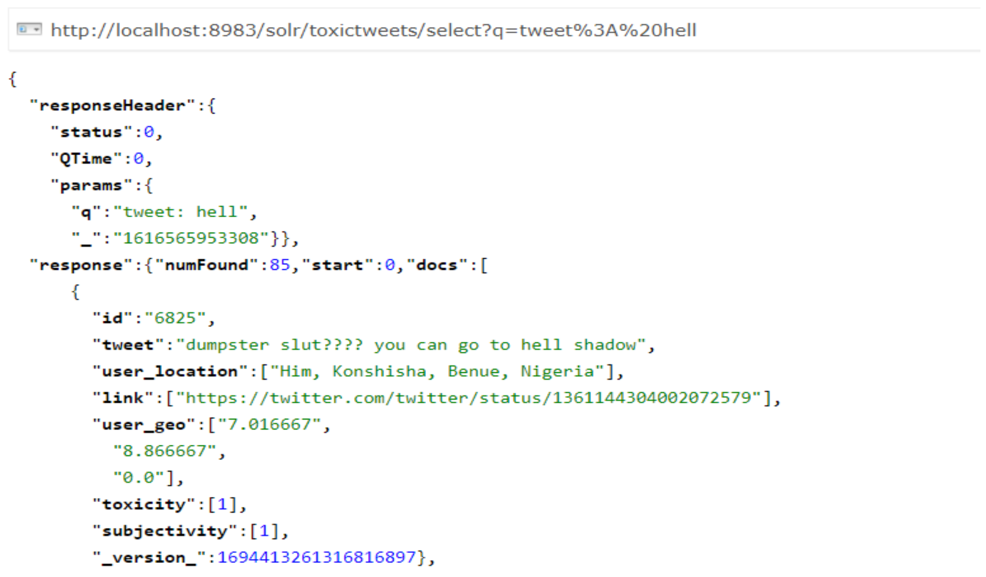
```
{
  "id":"0",
  "tweet":"RT @TrumpWarRoom___: RT to wish a Happy President's Day to Donald Trump.\r\n\r\nFollow @realGOPsupport",
  "user_location":["Virginia, United States"],
  "link":["https://twitter.com/twitter/status/1361390039108907008"],
  "user_geo":["37.1232245",
    "-78.4927721",
    "0.0"],
  "toxicity":[0],
  "subjectivity":[1],
  "_version_":1693916005106974720},
```

Figure 5: Sample of Data Indexing

3.1.4 Querying in Solr

After the data has been indexed successfully in Solr, we can perform any search query to retrieve the documents. As we have mentioned earlier, Solr allows communications through its REST APIs, including those for search queries. For our project, we will make search queries to Solr through GET requests. The following examples show sample queries and their results in Solr:

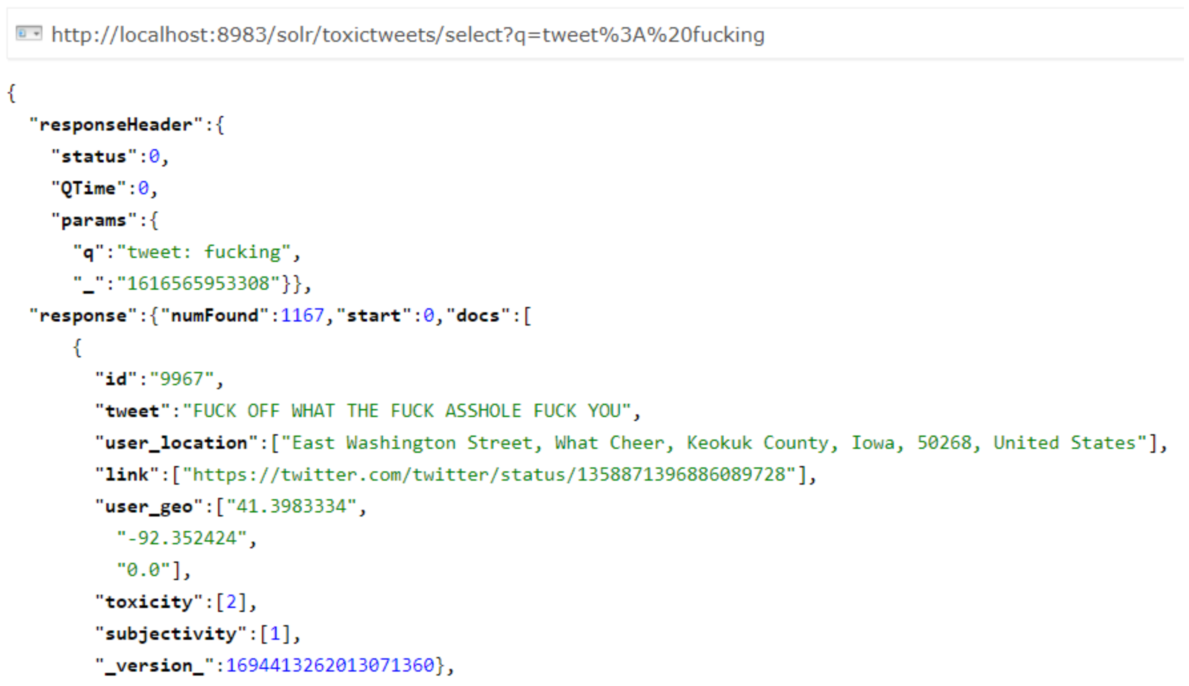
- In the first example, we perform a simple query of the word "hell" and Solr will return all documents (tweets) that contain the query word.



```
http://localhost:8983/solr/toxictweets/select?q=tweet%3A%20hell

{
  "responseHeader":{
    "status":0,
    "QTime":0,
    "params":{
      "q":"tweet: hell",
      "_":"1616565953308"}},
  "response":{"numFound":85,"start":0,"docs":[
      {
        "id":"6825",
        "tweet":"dumpster slut???? you can go to hell shadow",
        "user_location":["Him, Konshisha, Benue, Nigeria"],
        "link":["https://twitter.com/twitter/status/1361144304002072579"],
        "user_geo":["7.016667",
          "8.866667",
          "0.0"],
        "toxicity":[1],
        "subjectivity":[1],
        "_version_":1694413261316816897},
```

Figure 6: Query "hell" Results

- In the second example, we perform a query of the word “fucking” and Solr will return all documents (tweets) that contain the query word. Furthermore, as a result of stemming, Solr will also return relevant documents that contain the word “fuck”.

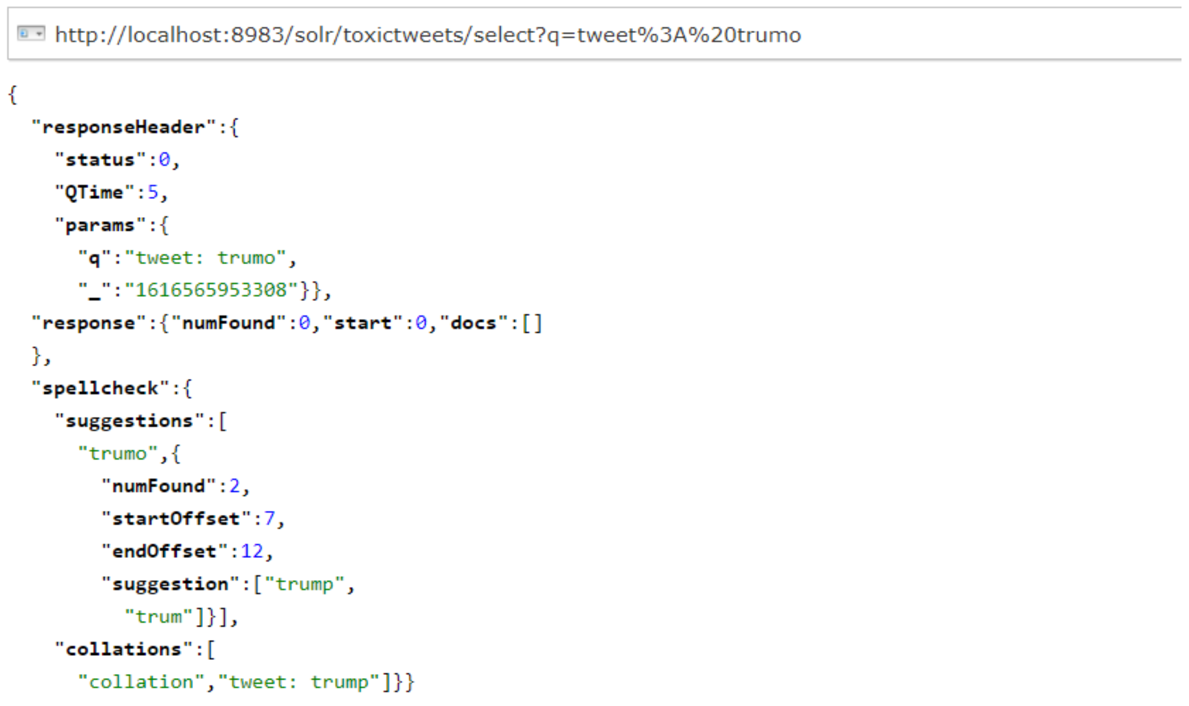


```
http://localhost:8983/solr/toxictweets/select?q=tweet%3A%20fucking
```

```
{
  "responseHeader":{
    "status":0,
    "QTime":0,
    "params":{
      "q":"tweet: fucking",
      "_":"1616565953308"}},
  "response":{"numFound":1167,"start":0,"docs":[
      {
        "id":"9967",
        "tweet":"FUCK OFF WHAT THE FUCK ASSHOLE FUCK YOU",
        "user_location":["East Washington Street, What Cheer, Keokuk County, Iowa, 50268, United States"],
        "link":["https://twitter.com/twitter/status/1358871396886089728"],
        "user_geo":["41.3983334",
          "-92.352424",
          "0.0"],
        "toxicity":[2],
        "subjectivity":[1],
        "_version_":1694413262013071360},
```

Figure 7: Query “fucking” Results

- In the last example we perform a typo query “trumo”. As a result of the typo, Solr returns no matching documents but it will do the spell checking and provide a suggestion “trump”.



```
http://localhost:8983/solr/toxictweets/select?q=tweet%3A%20trumo
```

```
{
  "responseHeader":{
    "status":0,
    "QTime":5,
    "params":{
      "q":"tweet: trumo",
      "_":"1616565953308"}},
  "response":{"numFound":0,"start":0,"docs":[]
  },
  "spellcheck":{
    "suggestions":[
      "trumo",{
        "numFound":2,
        "startOffset":7,
        "endOffset":12,
        "suggestion":["trump",
          "trum"]}],
    "collations":[
      "collation","tweet: trump"]}}
```

Figure 8: Spell Checking of Typo “trumo”

3.2 User Interface (UI) for Querying of Crawled Data

<u>Question 2: Build a simple web interface for the search engine (e.g., Google)</u>

The UI used to input and display queries from crawled data is a simple web application built using ReactJS. Aside from basic functions of querying and displaying results, the application also displays the results in a map and pie charts / Doughnuts as a form of innovation from the regular search.

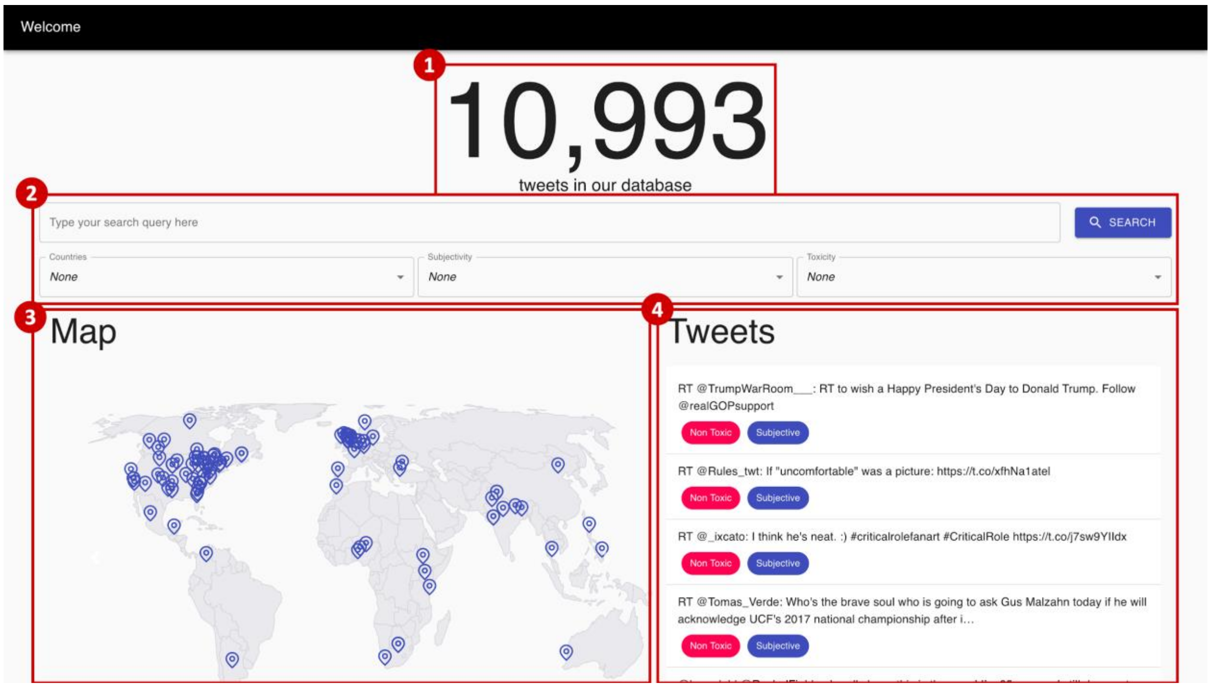
3.2.1 Tools

The ReactJS framework is used to create the UI of the application. It handles all frontend logic and makes HTTP calls to the backend. The framework uses JavaScript language and has a Component-based Architecture.

The components itself are built on top of the Material UI library's components. Some examples of the components being used are, FormControl for inputting queries, and Dialog for displaying query suggestions. The map component is created using the react-simple-maps API. The slideshow component that enables the user to cycle between the Map and the Charts uses react-bootstrap's Carousel. The pie chart itself uses react-chartjs-2's Doughnut component.

3.2.2 Design

Figure 1 shows the web application interface. The main functions of the application include querying and displaying tweets, filtering based on a country, subjectivity, or toxicity level, displaying the location of the tweets in a map, summarizing the tweet categories in a Doughnut, as well as summarizing the tweet statistics in a specific country using a Doughnut.



RT @TrumpWarRoom___: RT to wish a Happy President's Day to Donald Trump. Follow @realGOPsupport

Non Toxic Subjective

RT @Rules_twt: If "uncomfortable" was a picture: https://t.co/xfhNa1atel

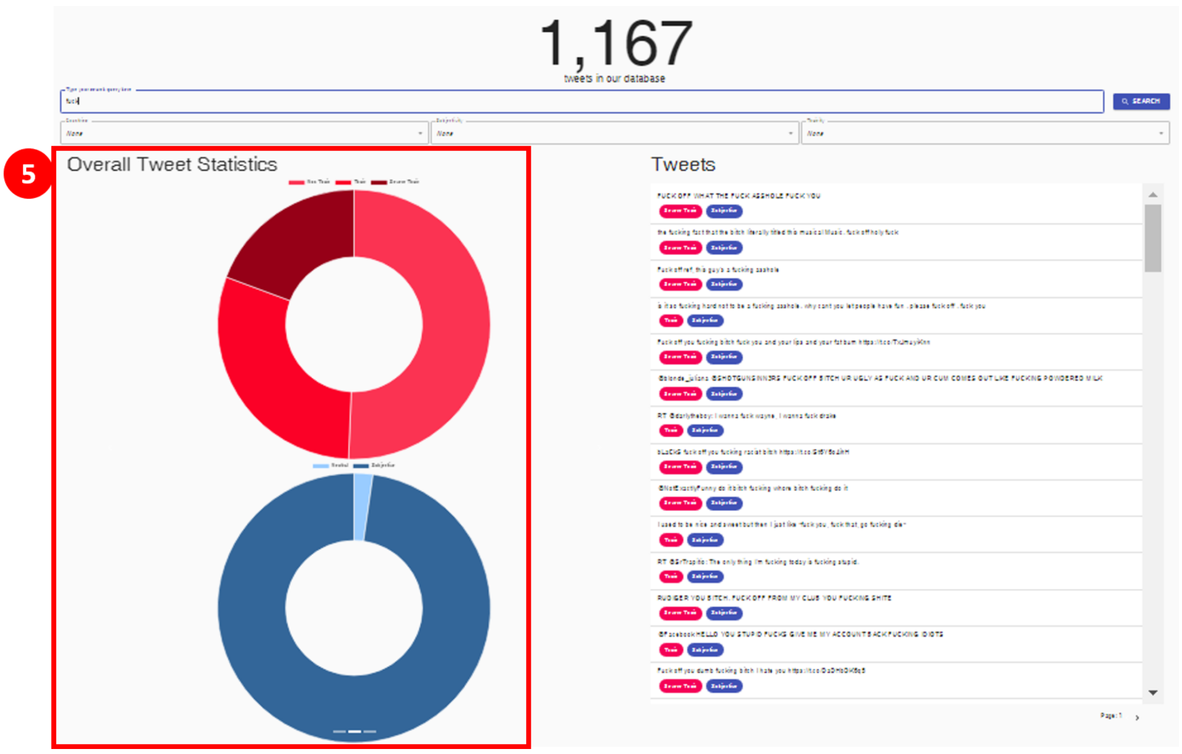
Non Toxic Subjective

RT @_ixcato: I think he's neat. :) #criticalrolefanart #CriticalRole https://t.co/j7sw9YIldx

Non Toxic Subjective

RT @Tomas_Verde: Who's the brave soul who is going to ask Gus Malzahn today if he will acknowledge UCF's 2017 national championship after i…

Non Toxic Subjective


1,167

tweets in our database

Type your search query here

fuck

SEARCH

Countries: None | Subjectivity: None | Toxicity: None

5 Overall Tweet Statistics

Tweets

FUCK OFF WHAT THE FUCK ASSHOLE FUCK YOU
Severe Toxic | Subjective

the fucking fact that the bitch literally titled this musical Music. fuck off holy fuck
Severe Toxic | Subjective

Fuck off ref, this guy's a fucking asshole
Severe Toxic | Subjective

is it so fucking hard not to be a fucking asshole . why cant you let people have fun . please fuck off . fuck you
Toxic | Subjective

Fuck off you fucking bitch fuck you and your lips and your fat bum https://t.co/TxJmayiKnn
Severe Toxic | Subjective

@blonde_juliana @SHOTGUNSINN3RS FUCK OFF BITCH UR UGLY AS FUCK AND UR CUM COMES OUT LIKE FUCKING POWDERED MILK
Severe Toxic | Subjective

RT @darlytheboy: I wanna fuck wayne , I wanna fuck drake
Toxic | Subjective

bLaCkG fuck off you fucking racist bitch https://t.co/Gt6Y6oJkH
Severe Toxic | Subjective

@NotExactlyFunny do it bitch fucking whore bitch fucking do it
Severe Toxic | Subjective

I used to be nice and sweet but then I just like "fuck you, fuck that, go fucking die"
Toxic | Subjective

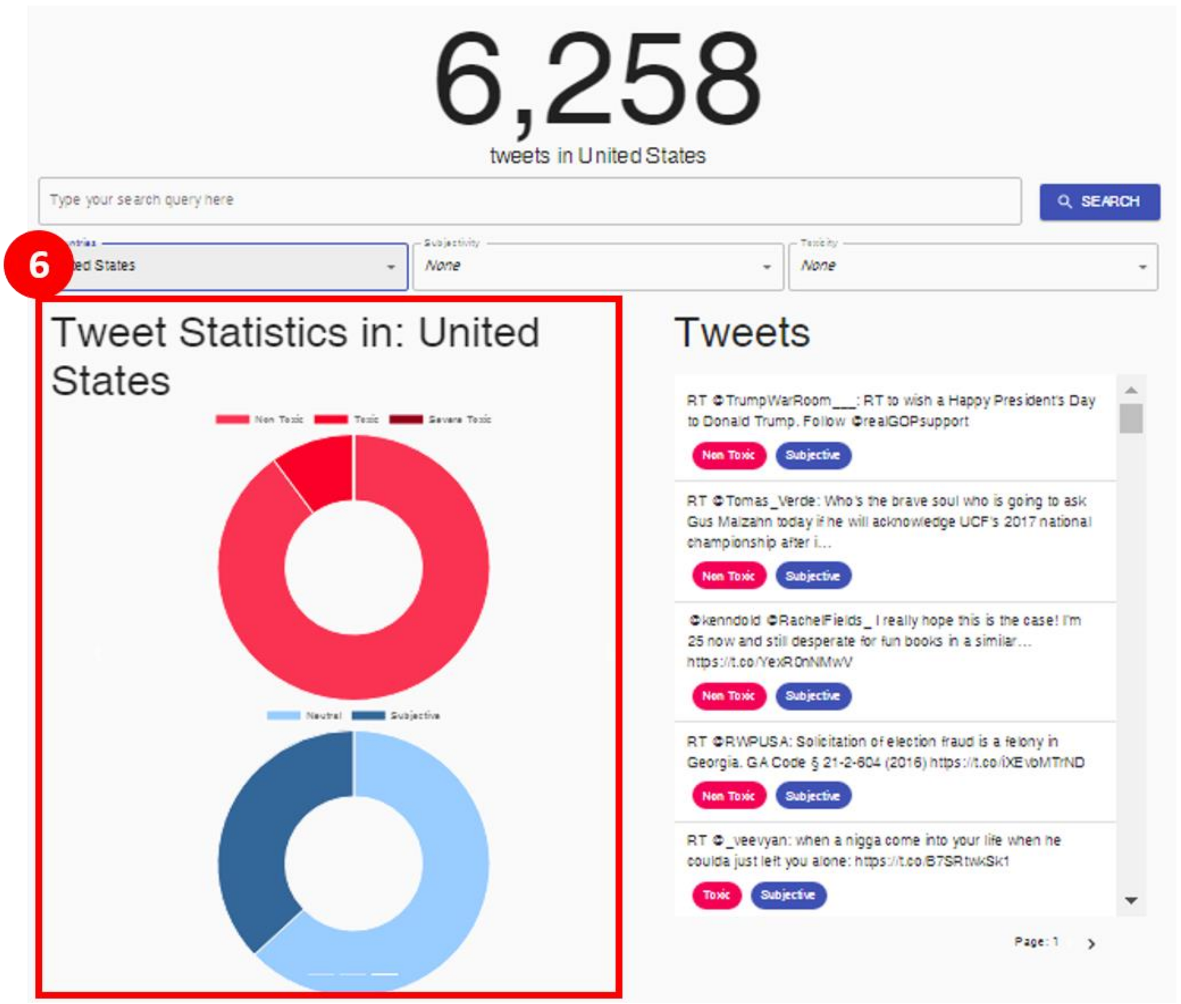
RT @SrTrapito: The only thing I'm fucking today is fucking stupid.
Toxic | Subjective

RUDIGER YOU BITCH. FUCK OFF FROM MY CLUB YOU FUCKING SHITE
Severe Toxic | Subjective

@Facebook HELLO YOU STUPID FUCKS GIVE ME MY ACCOUNT BACK FUCKING IDIOTS
Toxic | Subjective

Fuck off you dumb fucking bitch I hate you https://t.co/DaDHbOK6q5
Severe Toxic | Subjective

Page: 1 ›


6,258

tweets in United States

Type your search query here

SEARCH

6 Countries: United States | Subjectivity: None | Toxicity: None

Tweet Statistics in: United States

Tweets

RT @TrumpWarRoom___: RT to wish a Happy President's Day to Donald Trump. Follow @realGOPsupport
Non Toxic | Subjective

RT @Tomas_Verde: Who's the brave soul who is going to ask Gus Malzahn today if he will acknowledge UCF's 2017 national championship after i…
Non Toxic | Subjective

@kenndoid @RachelFields_ I really hope this is the case! I'm 25 now and still desperate for fun books in a similar… https://t.co/YexR0nNMwV
Non Toxic | Subjective

RT @RWPUSA: Solicitation of election fraud is a felony in Georgia. GA Code § 21-2-604 (2016) https://t.co/iXEvbMTrND
Non Toxic | Subjective

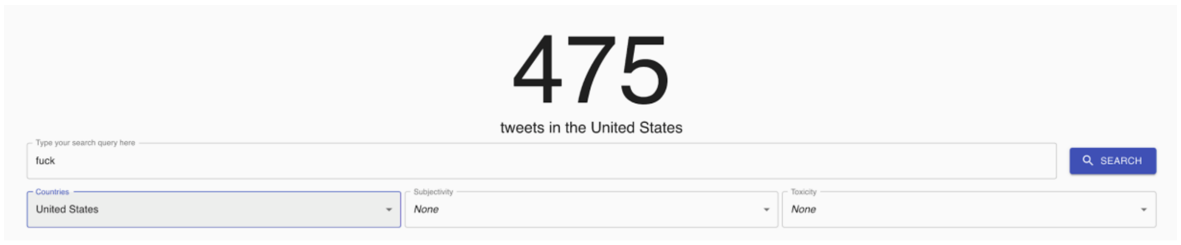
RT @_veevyan: when a nigga come into your life when he coulda just left you alone: https://t.co/B7SRtwkSk1
Toxic | Subjective

Page: 1 ›

Figure 9: Web Application Interface

1. Tweet Count

The tweet count component displays the number of tweets available in the database for a specific query. A sample of how the tweet count changes with changing filters is shown in the tables below.



475

tweets in the United States

Type your search query here: fuck

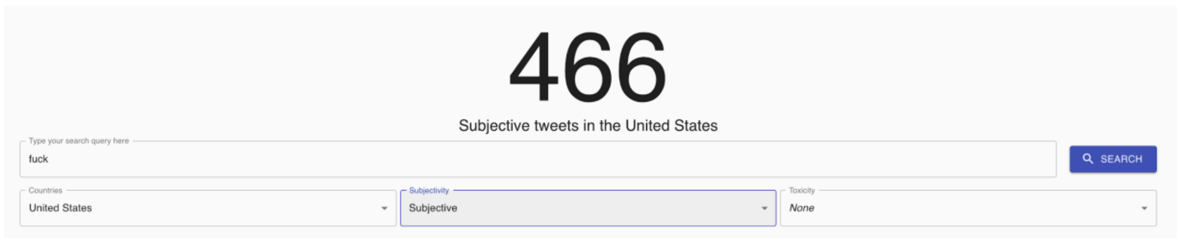
SEARCH

Countries: United States

Subjectivity: None

Toxicity: None

Query	Country	Subjectivity	Toxicity
fuck	United States	None	None



466

Subjective tweets in the United States

Type your search query here: fuck

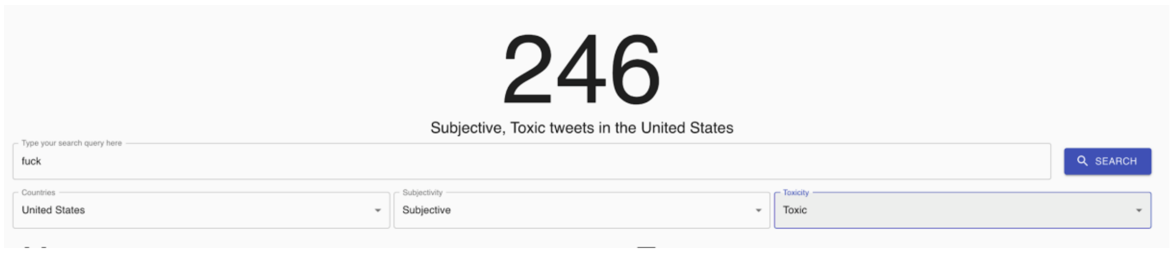
SEARCH

Countries: United States

Subjectivity: Subjective

Toxicity: None

Query	Country	Subjectivity	Toxicity
fuck	United States	Subjective	None



246

Subjective, Toxic tweets in the United States

Type your search query here: fuck

SEARCH

Countries: United States

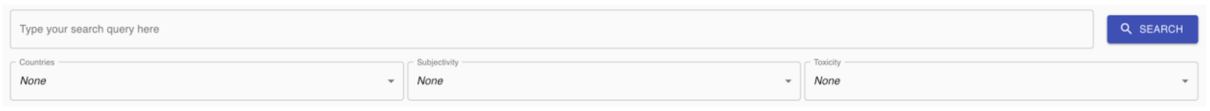
Subjectivity: Subjective

Toxicity: Toxic

Query	Country	Subjectivity	Toxicity
fuck	United States	Subjective	Toxic

Figure 10: Different Search Results with Filters Applied

2. Filter Form


Type your search query here

SEARCH

Countries: None

Subjectivity: None

Toxicity: None

Figure 11: Search Bar and Search Filters

The user can filter the tweets in the database based on search word, countries where the tweets originated from, the subjectivity of the tweet and the toxicity of the tweets. The filters for countries, subjectivity and toxicity are implemented in the form of a dropdown. The values for each filter are provided in the table below.

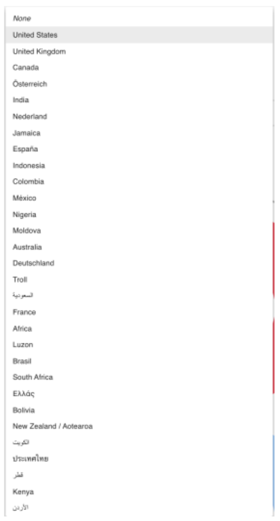
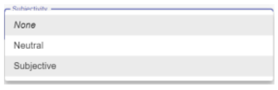
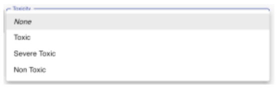
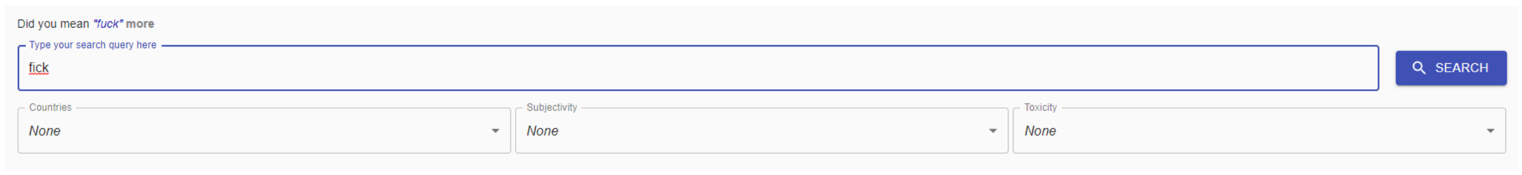
	Countries	**Subjectivity**	**Toxicity**
Number of Options	118	3	4
Filter Options	 *None* United States United Kingdom Canada Österreich India Nederland Jamaica España Indonesia Colombia México Nigeria Moldova Australia Deutschland Troll السعودية France Africa Luzon Brasil South Africa Ελλάς Bolivia New Zealand / Aotearoa الكويت ประเทศไทย قطر Kenya الأردن	 Subjectivity *None* Neutral Subjective	 Toxicity *None* Toxic Severe Toxic Non Toxic

Table 2: Values for Each Filter Option

Filter **None** indicates that the query will not be filtered by that specific filter category.


Did you mean *"fuck"* more

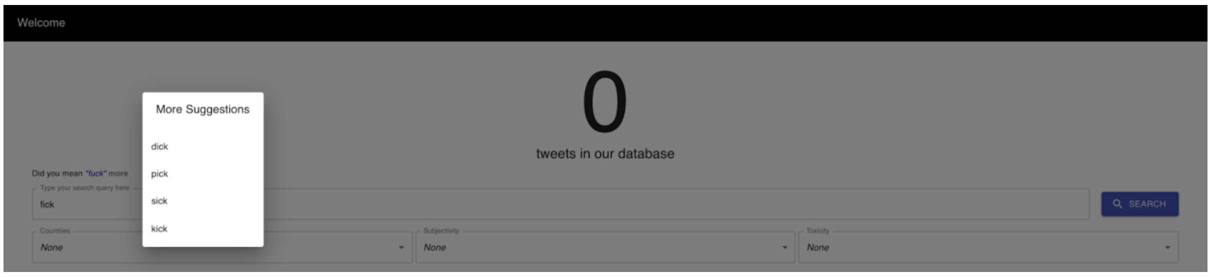
Type your search query here: fick

SEARCH

Countries: None

Subjectivity: None

Toxicity: None



Did you mean "fuck" more

Type your search query here: fick

Countries: None

Subjectivity: None

Toxicity: None

Figure 12: Query Suggestions for User Typo when Searching

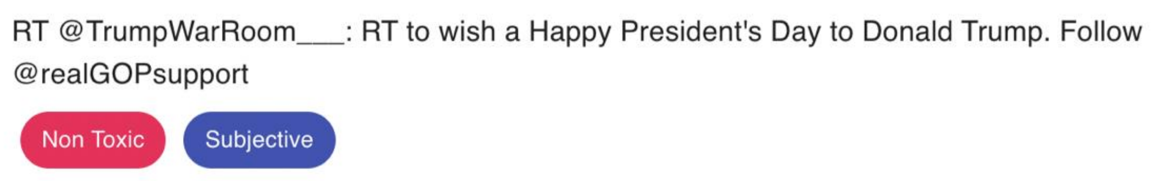
The two screenshots above show the app response when a query with a typo is searched. The app will suggest a top suggestion. Should this suggestion not be the user's intended query, the user can click *more* to display more query suggestions.

3. Map Display

The map displays the tweet origins. More explanation can be found in **Section 3.4.1**.

4. Tweet List

Below is an example of a tweet being displayed among the list of tweets.



RT @TrumpWarRoom___: RT to wish a Happy President's Day to Donald Trump. Follow @realGOPsupport

Figure 13: Sample Tweet Displayed by User Interface

Other than displaying the tweet contents, the app also provides labels on the tweet's subjectivity and toxicity classifications.

5. Tweet Statistics

The two Doughnuts show the distributions of a tweet's toxicity and subjectivity. More explanation can be found in **Section 3.4.2**.

6. Tweet Statistics in a Country

The distributions of a tweet's toxicity and subjectivity in a specific country are shown in the Doughnuts. More explanation can be found in **Section 3.4.2**.

3.3 Test Query Results and Performance

Question 2: Write five queries, get their results, and measure the speed of the querying

In this section, we write five test queries and illustrate their results. In detail, we use the Postman software to send test queries to the Solr search engine. For each query, Solr (backend) will return the query time (QTime) it needs to execute the query, whereas Postman (frontend) will calculate the total time (Time required) it requires to fully process the query. Their results and speed of querying (QTime and Time required) are measure and illustrated in the following table:

Id	**Query**	**Query filter**	**Number of results found**	**Top result**	**QTime (ms)**	**Time required (ms)**
1	hell	None	85	“dumpster slut???? you can go to hell shadow”	4	13
2	dumb	None	498	“@KaitOw0 Dummy dummy dummy dumb dumb”	2	8
3	loser	None	486	“You are a LOSER LOSER LOSER https://t.co/RE96gS8lVb https://t.co/udCzuF6eLY”	1	9
4	donald trump	None	118	“RT @ACTBrigitte: Donald J Trump is trending on Presidents Day for a reason!!”	6	19
5	corrupt	user_location: united states	22	“RT @depravedaddy: This dirty fucking pervert is aching for a young innocent little virgin whore to corrupt tonight.”	2	11

Table 3: 5 Sample Queries Results

3.4 Innovations for enhancing the indexing and ranking

Question 3: Explore some innovations for enhancing the indexing and ranking. Explain why they are important to solve specific problems, illustrated with examples.

In this section, we illustrate several innovative works that we have implemented in our toxic tweet search system. These innovations include geo-spatial search using map, enhanced search using doughnut charts, and search suggestion/spell checking.

3.4.1 Geo-spatial Search using Map

The map component pinpoints the location where a tweet was created. The default number of location pins displayed when no filter is chosen is 100. Otherwise, the map will display all locations of tweets being queried.

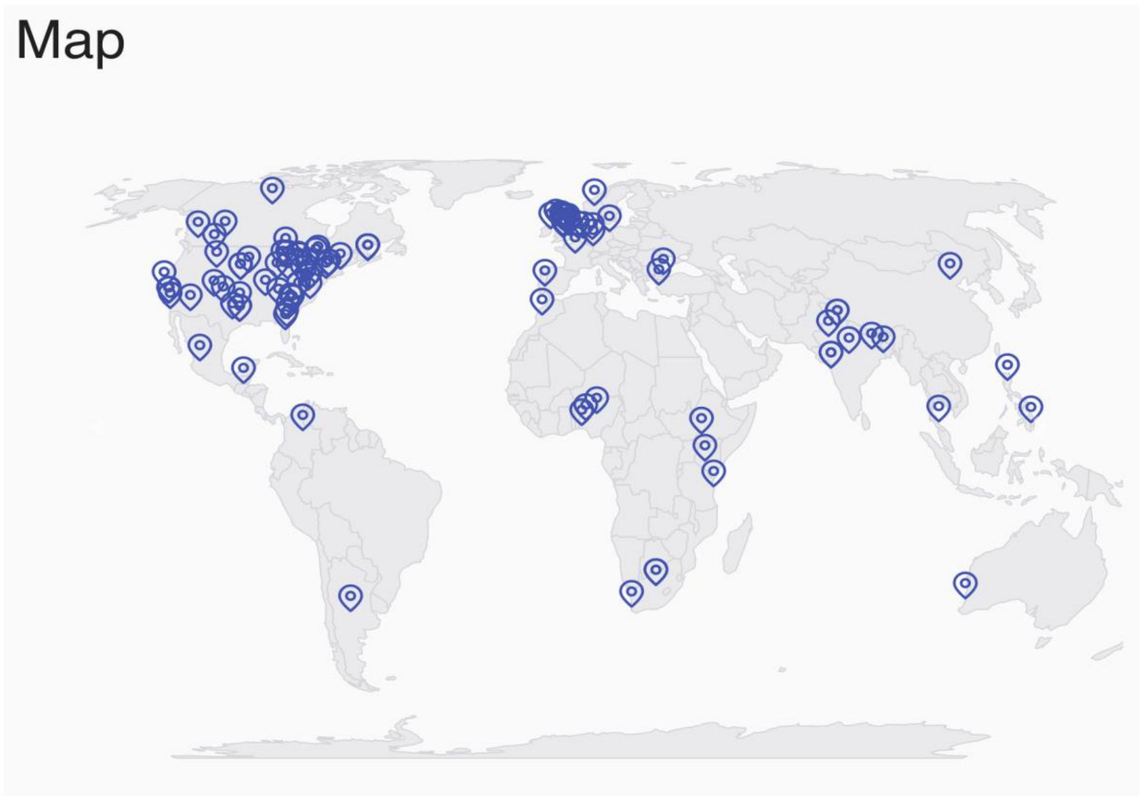


Figure 14: Geo-Spatial Display of Tweets Locations

3.4.2 Enhanced Search using Doughnut Charts

There are two implementations of the Enhanced Search innovation using Doughnut Charts. One is to display a tweet's classification distribution, and another is to display a country's classification distribution.

3.4.2.1 Tweet Statistics

This innovation displays the distribution of a query in terms of its toxicity and subjectivity using a Doughnut chart. The number of tweets that exist under a category will be displayed when a cursor hovers on the chart. For example, the figure below shows the Doughnut charts for when the term "die" is queried.

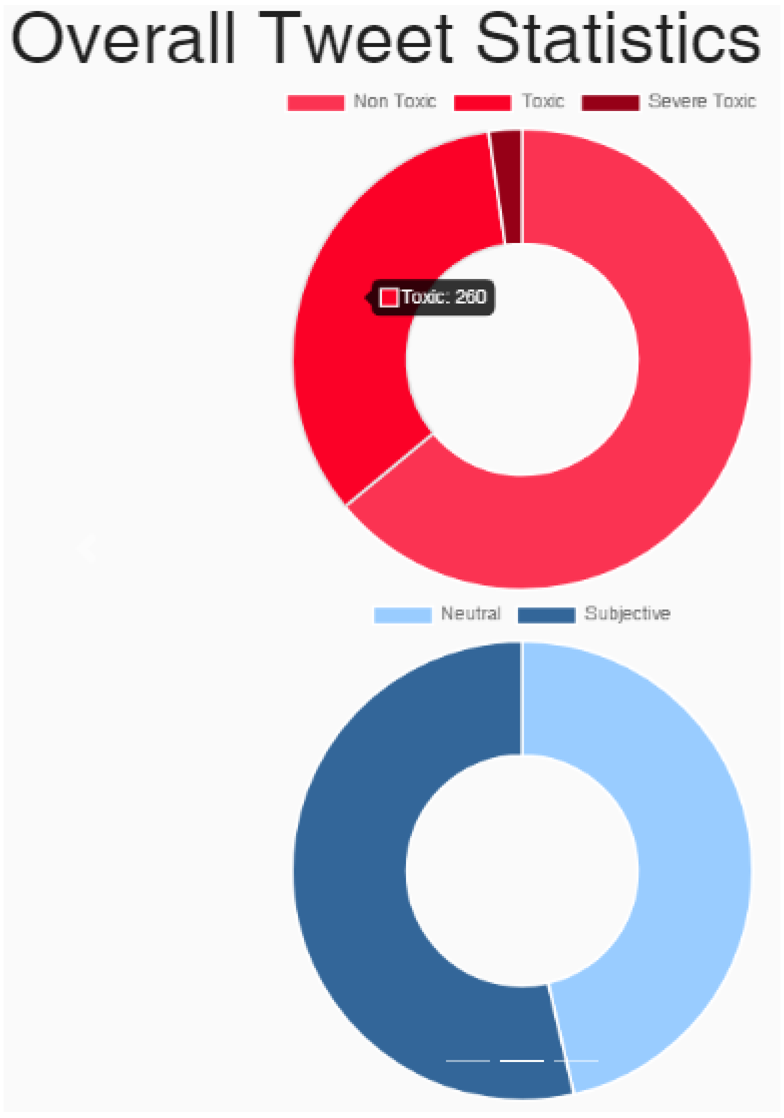


Figure 15: Doughnut Charts Display

3.4.2.2 Tweet Statistics in a Country

Likewise, this innovation displays a classification distribution in terms of its toxicity and subjectivity. One must select a country to be able to view the charts. The charts will show the distributions of a tweet in the chosen country should a query be inputted. The figure below shows the Doughnut charts for when Canada is chosen and "die" is queried.

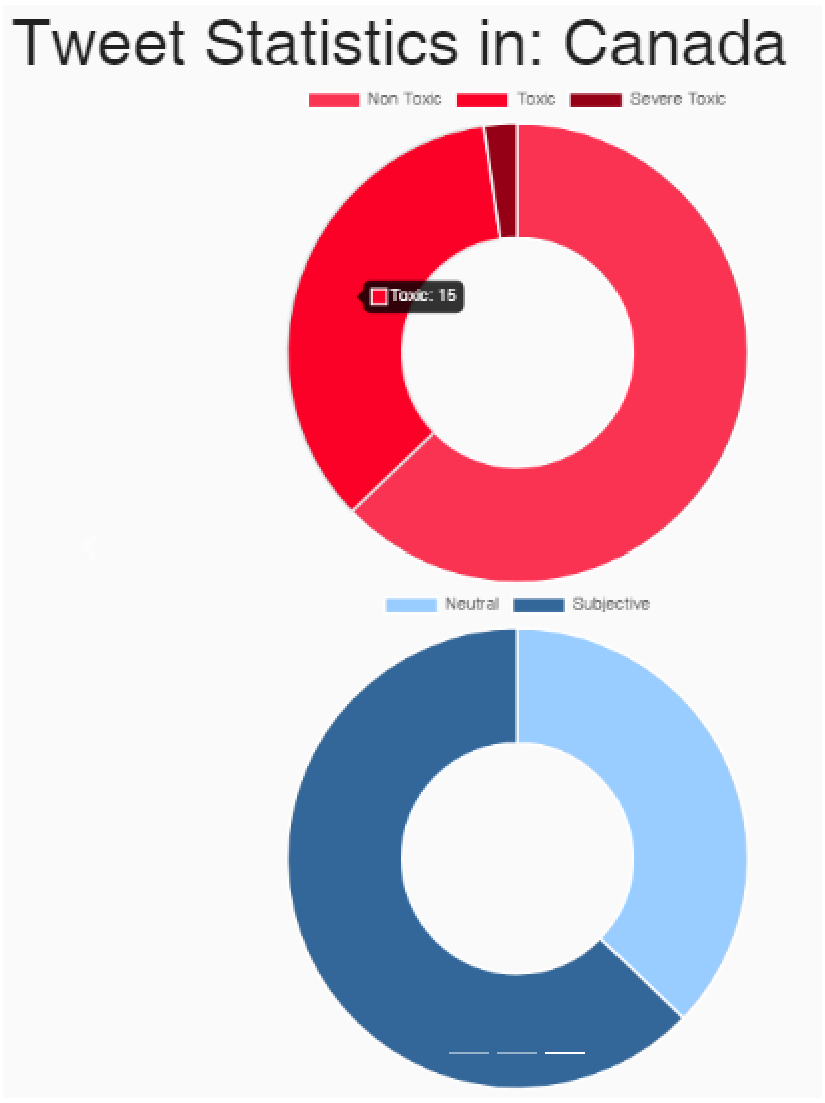
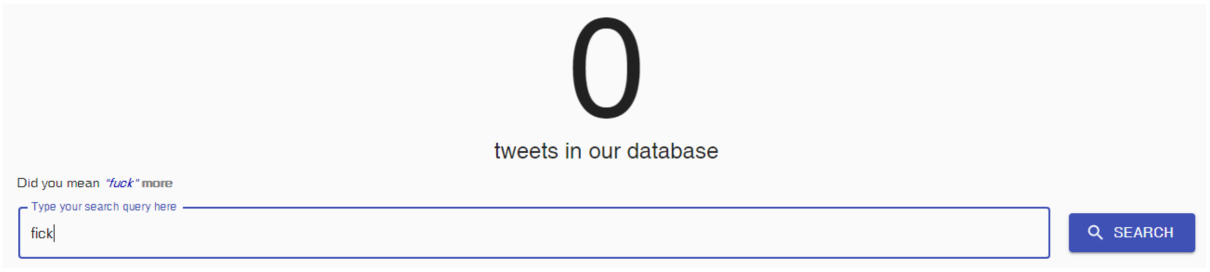


Figure 16: Doughnut Charts Display for Country Filter "Canada" and Query "die"

3.4.3 Search Suggestion and Spell Checking

The spell checking and search suggestion component provides suggestions to users when their queries return no matching documents. Users can choose to use the system's suggestions by clicking on it. Additionally, since Solr provides suggestions based on the created index and words in the lowercase form when the index is built, all suggestions are in lowercase form as well. The following example shows search suggestions provided by our system.



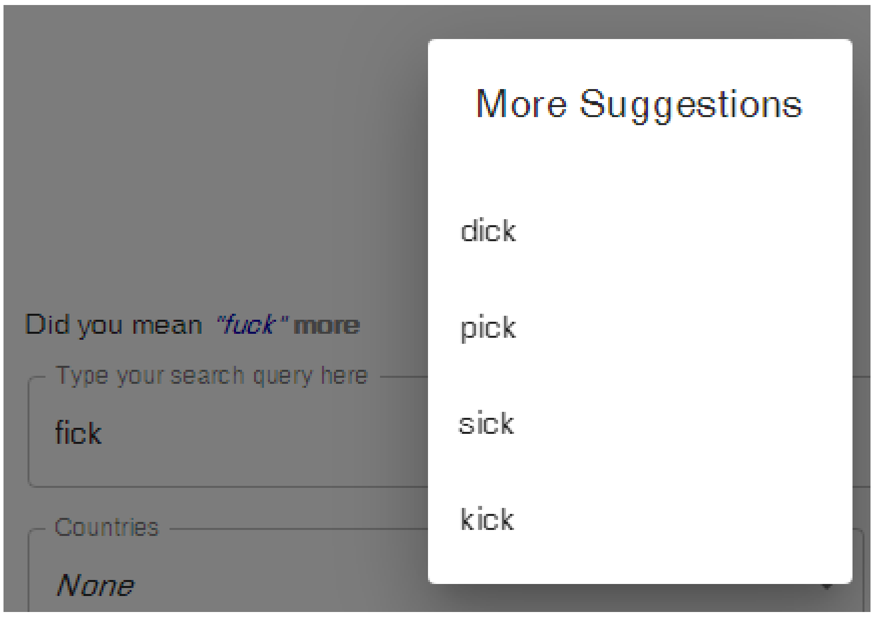


Figure 17: Search Suggestions

4. Classification

4.1 Approaches to Classification Problem

Question 4: Motivate the choice of your classification approach in relation with the state of the art

Our team decided to make use of State-of-the-Art Transformer-based machine learning models pre-trained on text data such as, Bidirectional Encoder Representations from Transformers (BERT) and Robustly Optimized BERT Pretraining Approach (RoBERTa), as our classification approach. Using these pre-trained models, we apply the use of Transfer learning approach in order to train the models to solve our Multitask Classification problem of subjectivity classification and toxicity classification.

Question 4: Discuss whether you had to preprocess data (e.g., microtext normalization) and why

Before we are able to perform classification on the crawled text data from Twitter, preprocessing must be done on each of the tweets to bring them into a form that is predictable and analyzable for our classification task. This is due to the fact that Twitter text data are "noisy" in general, with hashtags, emojis, and shortened word forms often present in tweets. We have to preprocess and clean the Twitter text data in order transform them into a more standardized form for our Classifier so that it can learn the training data well without being affected by noise present within the data.

Question 4: Build an evaluation dataset by manually labeling 10% of the collected data (at least 1,000 records) with an inter-annotator agreement of at least 80%

From the text corpus that we have crawled, we first manually labelled 1000 text data for our validation dataset, in addition, our team also manually labeled an additional 1759 texts to be used as training data to train the classification models.

Question 4: Provide evaluation metrics such as precision, recall, and F-measure and discuss results & Discuss performance metrics, e.g., records classified per second, and scalability of the system

In order to evaluate how well the two classifiers are able to classify the text data for subjectivity classification and toxicity classification, we will be using F1 score, precision score and recall score to evaluate the ability of the classifiers to make predictions for each of the class labels.

In addition to evaluating the ability of the classifiers to make predictions for each of the class labels, we will also be evaluating the speed of the classifiers in making predictions on records in terms of records classified per second.

Question 5: Explore some innovations for enhancing classification

In order to enhance the classification of the topic of hate speech and toxic comments on Twitter, our team has decided to tackle the classification of the topic of hate speech and toxic comments on Twitter as a multitask classification problem, which firstly classifies if a tweet is subjective or objective, and at the same time, classify the tweet based on its level of toxicity.

Also, in order to improve the prediction accuracy and performance of the classification model, our team explored both the use of an ensemble classifier as well as a two-headed classifier to perform the multitask classification, the model will consist of two State-of-the-Art Transformer-based models concatenated together to perform the classification task.

4.2 Classification Task

Our classification task consists of subjectivity classification and toxicity classification. The subjectivity classification task classifies a tweet as 1 if the tweet contains some sentiment feeling and 0 otherwise. For the toxicity classification, the label space contains three labels: 0, 1, 2. Label 0 indicates that the corresponding tweet is non-toxic while the other two labels 1 and 2 indicate the toxicity level of the tweet, namely, “toxic” or “severe toxic”, respectively. For the classification task, the training dataset that we use is the Kaggle toxic comments dataset combined with 1759 manually labeled tweets from our crawled dataset. The distribution of our combined training set is shown in the following figures:

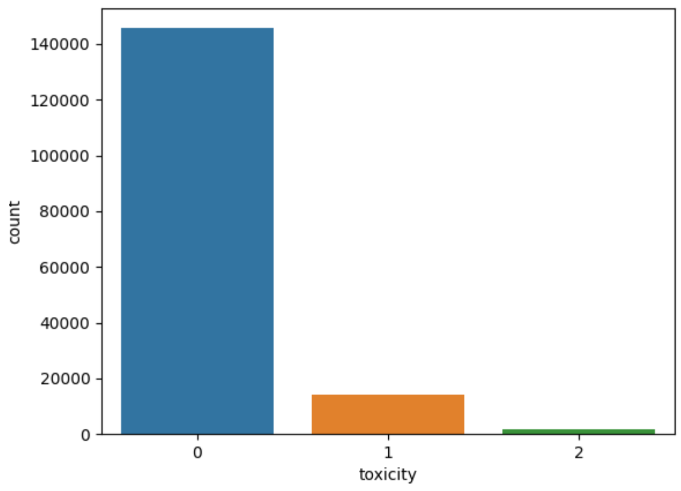


Figure 18: Distribution of toxic labels

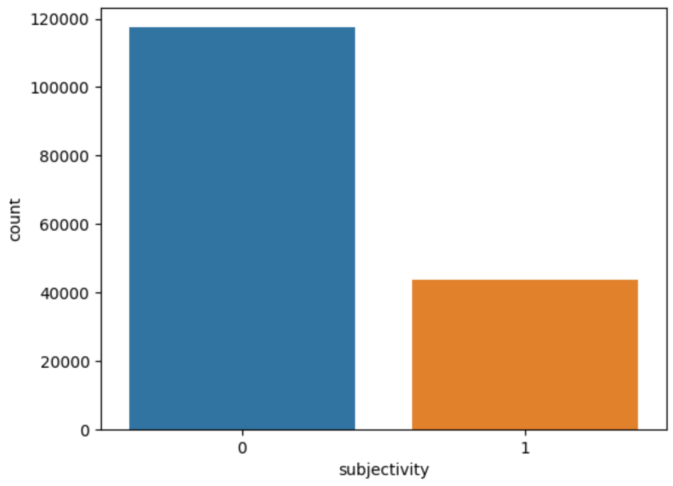


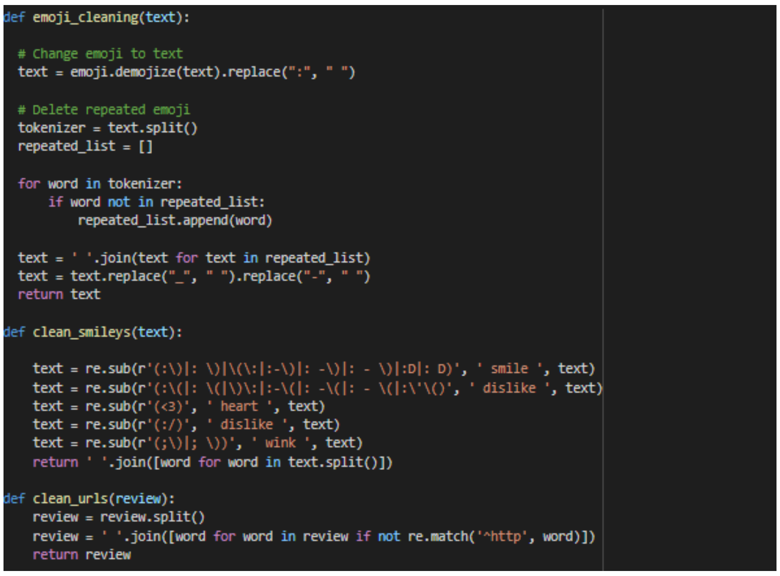
Figure 19: Distribution of subjectivity labels

4.3 Preprocessing of Tweets for Classification

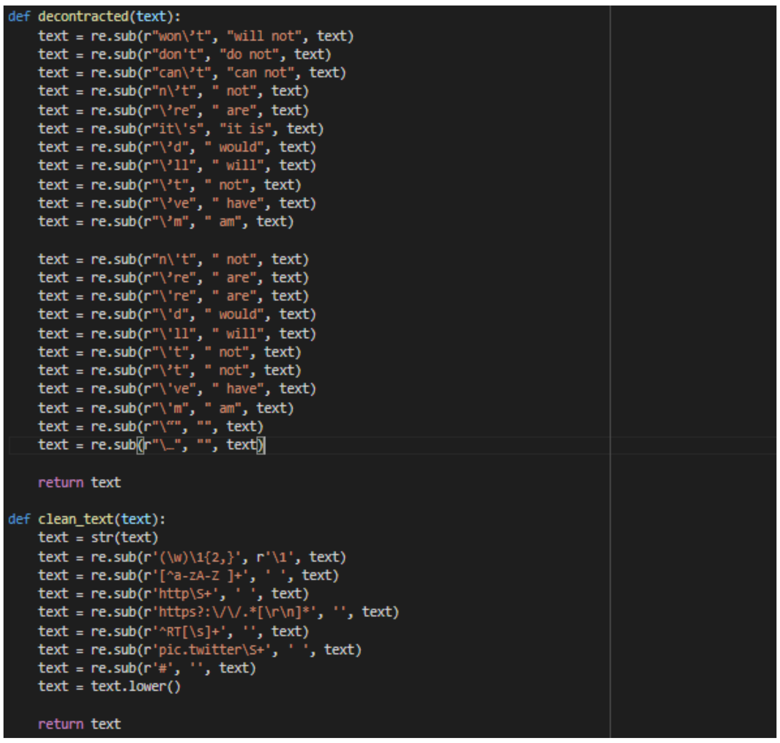
Before we are able to perform classification on the crawled text data from Twitter, preprocessing must be done on each of the tweets to bring them into a form that is predictable and analyzable for our classification task.

This is due to the fact that Twitter text data are “noisy” in general, with hashtags, emojis, and shortened word forms often present in tweets. We have to preprocess and clean the Twitter text data in order transform them into a more standardized form for our Classifier so that it can learn the training data well without being affected by noise present within the data.

4.2.1 Preprocessing Functions



```python
def emoji_cleaning(text):

  # Change emoji to text
  text = emoji.demojize(text).replace(":", " ")

  # Delete repeated emoji
  tokenizer = text.split()
  repeated_list = []

  for word in tokenizer:
      if word not in repeated_list:
          repeated_list.append(word)

  text = ' '.join(text for text in repeated_list)
  text = text.replace("_", " ").replace("-", " ")
  return text

def clean_smileys(text):

    text = re.sub(r'(:\)|: \)|\(\:|:-\)|: -\)|: - \)|:D|: D)', ' smile ', text)
    text = re.sub(r'(:\(|: \(|\)\:|:-\(|: -\(|: - \(|:\'\()', ' dislike ', text)
    text = re.sub(r'(<3)', ' heart ', text)
    text = re.sub(r'(:/)', ' dislike ', text)
    text = re.sub(r'(;\)|; \))', ' wink ', text)
    return ' '.join([word for word in text.split()])

def clean_urls(review):
    review = review.split()
    review = ' '.join([word for word in review if not re.match('^http', word)])
    return review
```



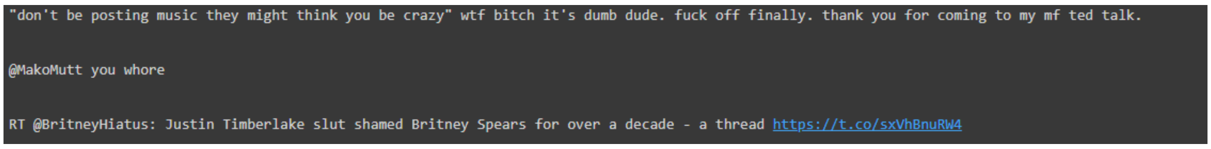
```python
def decontracted(text):
    text = re.sub(r"won\’t", "will not", text)
    text = re.sub(r"don't", "do not", text)
    text = re.sub(r"can\’t", "can not", text)
    text = re.sub(r"n\’t", " not", text)
    text = re.sub(r"\’re", " are", text)
    text = re.sub(r"it\'s", "it is", text)
    text = re.sub(r"\’d", " would", text)
    text = re.sub(r"\’ll", " will", text)
    text = re.sub(r"\’t", " not", text)
    text = re.sub(r"\’ve", " have", text)
    text = re.sub(r"\’m", " am", text)

    text = re.sub(r"n\'t", " not", text)
    text = re.sub(r"\’re", " are", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\’t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)
    text = re.sub(r"\“", "", text)
    text = re.sub(r"\_", "", text)

    return text

def clean_text(text):
    text = str(text)
    text = re.sub(r'(\w)\1{2,}', r'\1', text)
    text = re.sub(r'[^a-zA-Z ]+', ' ', text)
    text = re.sub(r'http\S+', ' ', text)
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text)
    text = re.sub(r'^RT[\s]+', '', text)
    text = re.sub(r'pic.twitter\S+', ' ', text)
    text = re.sub(r'#', '', text)
    text = text.lower()

    return text
```

Figure 20: Code Snippet of Functions to Preprocess Tweet Text

In the figures above are the preprocessing functions that have been defined by us for the purpose of preprocessing the Twitter text data. The emoji_cleaning() and clean_smileys() functions are used to clean and remove any emojis or emoticons present in the text data. Next, the clean_urls() function is used to remove any url links contained within a tweet so that they do not affect the training of the classifier as most urls do not contain information relevant to the text itself. Finally, the decontracted() and clean_text() functions are to recover any shortened word forms into proper word form and remove any other noise in the tweets such as hashtags, retweet tags, and as well as to convert the tweets into lower case form. When all these functions are applied to the tweets in our crawled text corpus, the tweets will be in a more standardized format to be able to be used for training our Classifier. We do not remove any stop words in the text data as we will be making use of State-of-the-Art Machine Learning models that are able to handle common English stop words without affecting their prediction accuracy. Tokenization of text data will be done in the classification step of the classification pipeline.

4.2.2 Processed and Cleaned Tweets

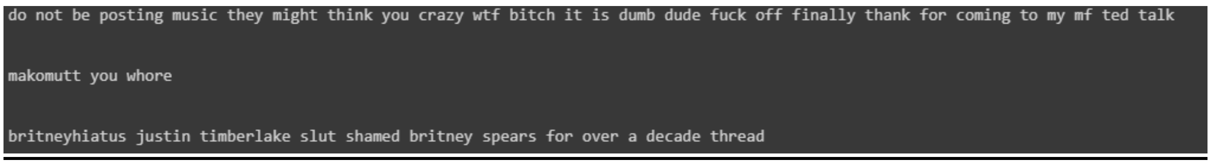
Before



```
"don't be posting music they might think you be crazy" wtf bitch it's dumb dude. fuck off finally. thank you for coming to my mf ted talk.

@MakoMutt you whore

RT @BritneyHiatus: Justin Timberlake slut shamed Britney Spears for over a decade - a thread https://t.co/sxVhBnuRW4
```

Figure 21: Tweet Texts Before Preprocessing

After



```
do not be posting music they might think you crazy wtf bitch it is dumb dude fuck off finally thank for coming to my mf ted talk

makomutt you whore

britneyhiatus justin timberlake slut shamed britney spears for over a decade thread
```

Figure 22: Tweet Texts After Preprocessing

In the figures above, we can observe 3 example tweets before and after they are processed by the preprocessing functions as defined in the section above. Before the tweets are processed, we can observe the presence of noise within the tweets such as punctuation marks, shortened word form, urls, as well as retweet tags. After the tweets are processed by the preprocessing functions, the noise within the tweets are removed and they are in a more analyzable and standardized format for our Classifier to train on.

4.4 Classification Approach

In order to tackle the Natural Language Processing (NLP) Multitask Classification problem that our team has defined, we decided to make use of State-of-the-Art Transformer-based machine learning models pre-trained on text data such as, Bidirectional Encoder Representations from Transformers (BERT) and Robustly Optimized BERT Pretraining Approach (RoBERTa), as our classification approach. Using these pre-trained models, we apply the use of Transfer learning approach in order to train the models to solve our Multitask Classification problem of subjectivity classification and toxicity classification.

The reason for using Transformer-based machine learning models is due to the fact that, similar to recurrent neural networks (RNNs), Transformers are designed to handle sequential data, such as natural language. In addition, unlike RNNs, Transformers do not require that the sequential data be processed in order. For example, if the input data is a natural language sentence, the Transformer does not need to process the beginning of it before the end which allows for more parallelization than RNNs and therefore reduced training times.

The use of Transfer Learning in our classification approach also helps to speed up training and improve the performance of our classification model. Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. By making use of Transfer Learning techniques on pre-trained NLP models such as BERT, we will be able to make use of pre-trained weights of the BERT model to learn the task of subjectivity and toxicity classification, which allows for shorter training times and avoid having to build a Transformer-based model from scratch which may not deliver good classification performance.

Our team also made use of pre-labeled text data from Kaggle, from the Toxic Comment Classification Challenge(https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data?select=train.csv.zip), in addition to the text data that was manually labeled by us from the text corpus that we have crawled in order to allow the classification models to have more text data to train on.

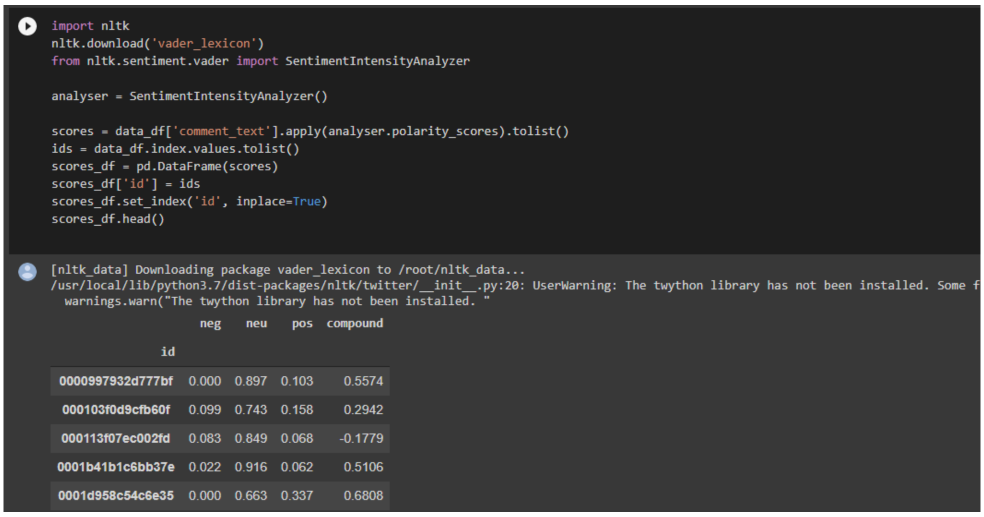
Also, in order to reduce the time required to train the classification models, we trained the models on the Google Colab platform in order to be able to make use of the Tensor Processing Unit (TPU) developed by Google when training our classification models. The reason for using the TPU accelerator for training our models is because training Transformer-based models on a large text corpus is computationally intensive, by utilizing the TPU accelerator, we are able achieve a significantly faster training time when training our models as compared to using a Graphics Processing Unit (GPU) accelerator or no hardware accelerator at all. In order to utilize the TPU accelerator in Google Colaboratory, we will be required to convert our training data into tf.data format as it is the only data format the TPU accelerator is able to process.

4.3.1 Training Data

Kaggle Dataset

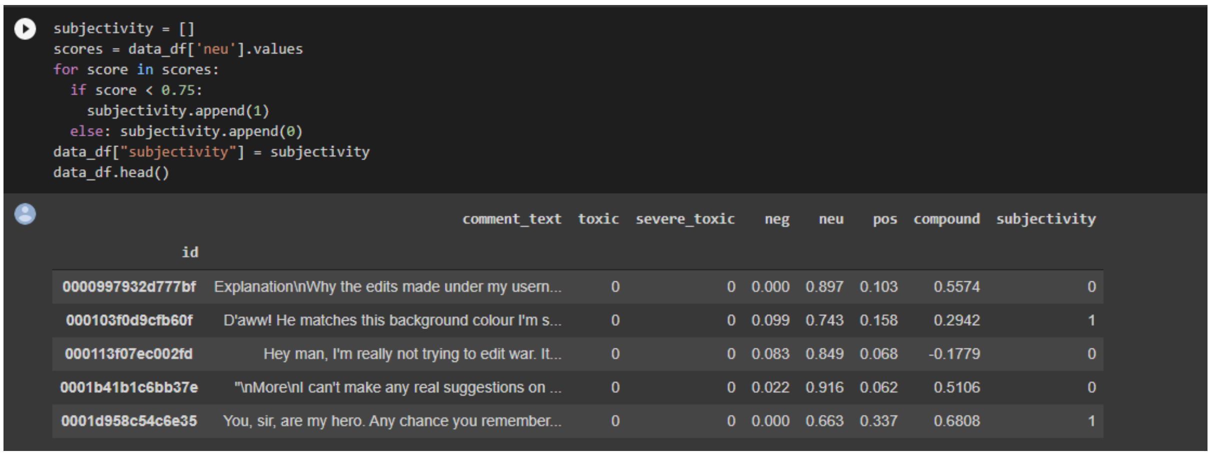
As it would be tedious to manually all the text data that we have crawled from Twitter, our team required a solution to reduce the need for manually labelling a huge portion of the crawled text corpus and yet provide the classification models with sufficient training data for them to learn the classification tasks that we have defined.

Therefore, in addition to manually labeled data from the crawled text corpus, our team also made use of the Toxic Comment Classification Challenge dataset from Kaggle for training our models. The dataset contains a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. The types of toxicity are: toxic, severe_toxic, obscene,threat, insult, identity_hate. For our classification task, we will only be making use of the toxic, severe_toxic of the Kaggle dataset for the toxicity classification where if both toxic, severe_toxic columns of the Kaggle dataset are 0, the text is non-toxic, if toxic column is 1 and severe_toxic column is 0, the text is toxic, if toxic column is 1 and severe_toxic column is 1, the text is severely toxic. For the subjectivity labels, we labeled those texts where if the toxic column is 1 or the severe_toxic column is 1, the text is subjective and labeled 1. For those text data with both toxic, severe_toxic columns of the Kaggle dataset are 0, we run the text through the nltk.sentiment.vader library's SentimentIntensityAnalyzer to get the neutral score of the texts. For those texts with a neutral score of above 0.75, they are labeled as 0 for non-subjective and those with a neutral score of below 0.75 they are labeled as 1 for subjective.



```python
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer

analyser = SentimentIntensityAnalyzer()

scores = data_df['comment_text'].apply(analyser.polarity_scores).tolist()
ids = data_df.index.values.tolist()
scores_df = pd.DataFrame(scores)
scores_df['id'] = ids
scores_df.set_index('id', inplace=True)
scores_df.head()
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
/usr/local/lib/python3.7/dist-packages/nltk/twitter/__init__.py:20: UserWarning: The twython library has not been installed. Some f
  warnings.warn("The twython library has not been installed. "
                    neg    neu    pos  compound
              id
0000997932d777bf  0.000  0.897  0.103    0.5574
000103f0d9cfb60f  0.099  0.743  0.158    0.2942
000113f07ec002fd  0.083  0.849  0.068   -0.1779
0001b41b1c6bb37e  0.022  0.916  0.062    0.5106
0001d958c54c6e35  0.000  0.663  0.337    0.6808
```



```python
subjectivity = []
scores = data_df['neu'].values
for score in scores:
  if score < 0.75:
    subjectivity.append(1)
  else: subjectivity.append(0)
data_df["subjectivity"] = subjectivity
data_df.head()
```

```
                                                       comment_text  toxic  severe_toxic    neg    neu    pos  compound  subjectivity
              id
0000997932d777bf  Explanation\nWhy the edits made under my usern...      0             0  0.000  0.897  0.103    0.5574             0
000103f0d9cfb60f  D'aww! He matches this background colour I'm s...      0             0  0.099  0.743  0.158    0.2942             1
000113f07ec002fd         Hey man, I'm really not trying to edit war. It...      0             0  0.083  0.849  0.068   -0.1779             0
0001b41b1c6bb37e    "\nMore\nI can't make any real suggestions on ...      0             0  0.022  0.916  0.062    0.5106             0
0001d958c54c6e35  You, sir, are my hero. Any chance you remember...      0             0  0.000  0.663  0.337    0.6808             1
```

Figure 23: Labeling of Kaggle Dataset for Subjectivity Classification Using nltk.sentiment.vader

Manually Labeled Data from Crawled Text Corpus

From the text corpus that we have crawled, in addition to the 1000 text data that is required to be manually labeled as our validation dataset, our team also manually labeled an additional 1759 texts to be used as training data to train the classification models.

Two-Step Classification Using Training Data

In order to train the classification models in the most efficient manner and to achieve the best possible classification accuracy for them, we will be training the classification models with the training data using a two steps approach.

First, the classification models will be trained on the Kaggle dataset from the Toxic Comment Classification Challenge to allow the Transformer-based pre-trained models to update their weights for the classification tasks of subjectivity and toxicity classification. The Kaggle dataset is used for the first step of training the classification models as it contains a large number of texts, with about 110,000 texts that can be used to train the models. The training is done using K-fold cross validation (4-fold) training to train each of the models which is achieved by running each model in a for loop to pre-process the data into tensorflow training and validation datasets and train the model for 4 epochs for each fold of the cross validation, for a total of 4 folds.

Next, after the classification models are trained on the Kaggle dataset, the weights of the models will be adjusted for the classification tasks of subjectivity and toxicity classification. Afterwhich, to further improve the prediction accuracy of the classification models for our crawled text corpus, the classification models are trained on the manually labeled additional 1759 texts from our crawled text corpus in order to fine-tune the models. The training is done using K-fold cross validation (4-fold) training to train each of the models which is achieved by running each model in a for loop to pre-process the data into Tensorflow training and validation datasets and train the model for 4 epochs for each fold of the cross validation, for a total of 4 folds.

Therefore, with this two steps classification approach, our models were able to achieve better classification accuracy as compared to just training the models purely on the Kaggle dataset or purely on the manually labeled additional 1759 texts from our crawled text corpus.

4.3.2 Ensemble Classifier

For our first approach, our team made use of ensemble learning to train our classifier. Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. For our ensemble learning, our team selected the Bidirectional Encoder Representations from Transformers (BERT) and Robustly Optimized BERT Pretraining Approach (RoBERTa) models for ensembling.

For each of the models, we obtain the model layers based on their Tensorflow implementation and added in some Dense Layers with ‘relu’ activation to be used to train on our training dataset. The figures below show the model architecture of the BERT and RoBERTa models.

<u>**BERT Model**</u>

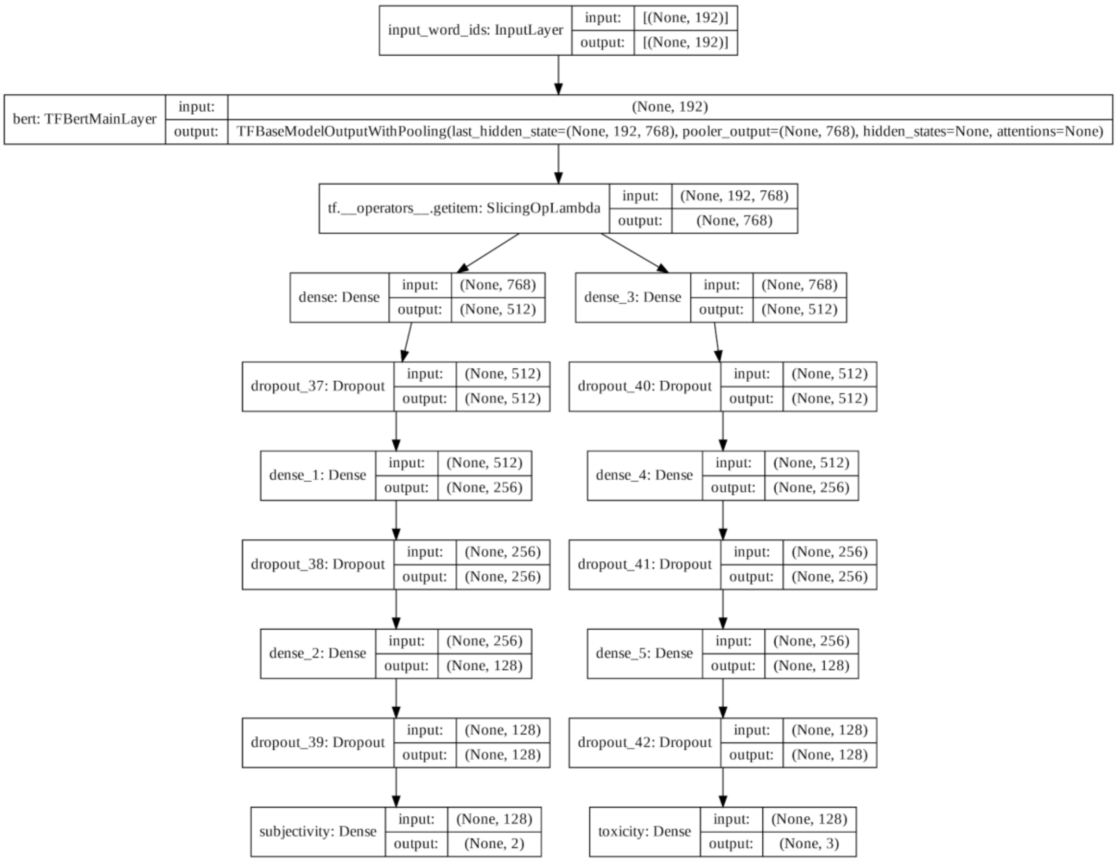
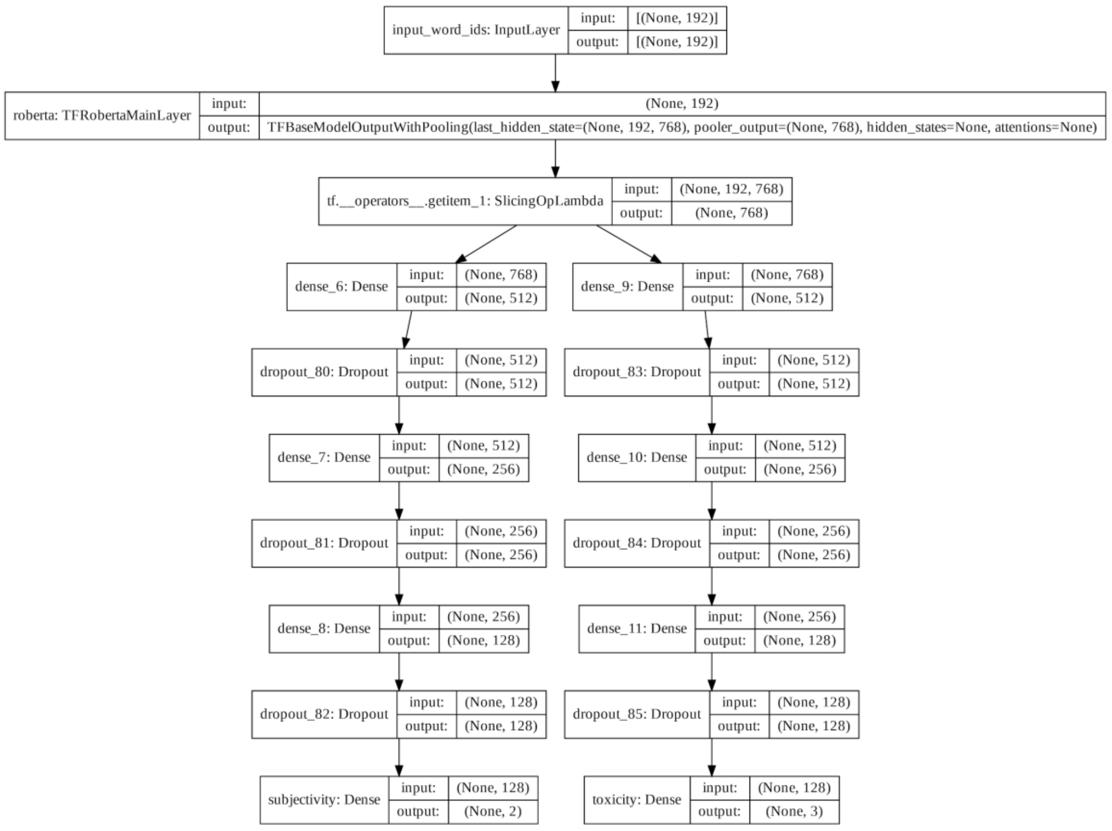


Figure 24: BERT Model Architecture

<u>RoBERTa Model</u>



input_word_ids: InputLayer | input: [(None, 192)] | output: [(None, 192)]
roberta: TFRobertaMainLayer | input: (None, 192) | output: TFBaseModelOutputWithPooling(last_hidden_state=(None, 192, 768), pooler_output=(None, 768), hidden_states=None, attentions=None)
tf.__operators__.getitem_1: SlicingOpLambda | input: (None, 192, 768) | output: (None, 768)
dense_6: Dense | input: (None, 768) | output: (None, 512)
dense_9: Dense | input: (None, 768) | output: (None, 512)
dropout_80: Dropout | input: (None, 512) | output: (None, 512)
dropout_83: Dropout | input: (None, 512) | output: (None, 512)
dense_7: Dense | input: (None, 512) | output: (None, 256)
dense_10: Dense | input: (None, 512) | output: (None, 256)
dropout_81: Dropout | input: (None, 256) | output: (None, 256)
dropout_84: Dropout | input: (None, 256) | output: (None, 256)
dense_8: Dense | input: (None, 256) | output: (None, 128)
dense_11: Dense | input: (None, 256) | output: (None, 128)
dropout_82: Dropout | input: (None, 128) | output: (None, 128)
dropout_85: Dropout | input: (None, 128) | output: (None, 128)
subjectivity: Dense | input: (None, 128) | output: (None, 2)
toxicity: Dense | input: (None, 128) | output: (None, 3)

Figure 25: RoBERTa Model Architecture

With the 2 models, we then make use of K-fold cross validation (4-fold) training to train each of the models which is achieved by running each model in a for loop to pre-process the data into Tensorflow training and validation datasets and train the model for 4 epochs for each fold of the cross validation, for a total of 4 folds.

After the 2 models are trained with the training data, we then use the trained models to make predictions on the test dataset. We then averaged the prediction probabilities from the 2 models, which is the simplest method of ensembling, to obtain the final prediction results.

4.3.3 Multi-Head Classifier

For our second approach, instead of having to train 2 models and ensemble their prediction results together, our team experimented with an alternative approach that will allow us to still make use of the two pre-trained transformer-based models, BERT and RoBERTa, for our multitask classification.

We developed a two-headed classifier to perform the multitask classification. The model will consist of two input heads, where each will be fed with text data tokenized by the BERT and RoBERTa tokenizers respectively. The two different sets of input data will then be passed into the respective BERT and

RoBERTa models which are then connected to a single Average( ) layer, to average the embeddings of the two models which are then used to generate the 2 outputs for subjectivity and toxicity classification.

Multi-head Model

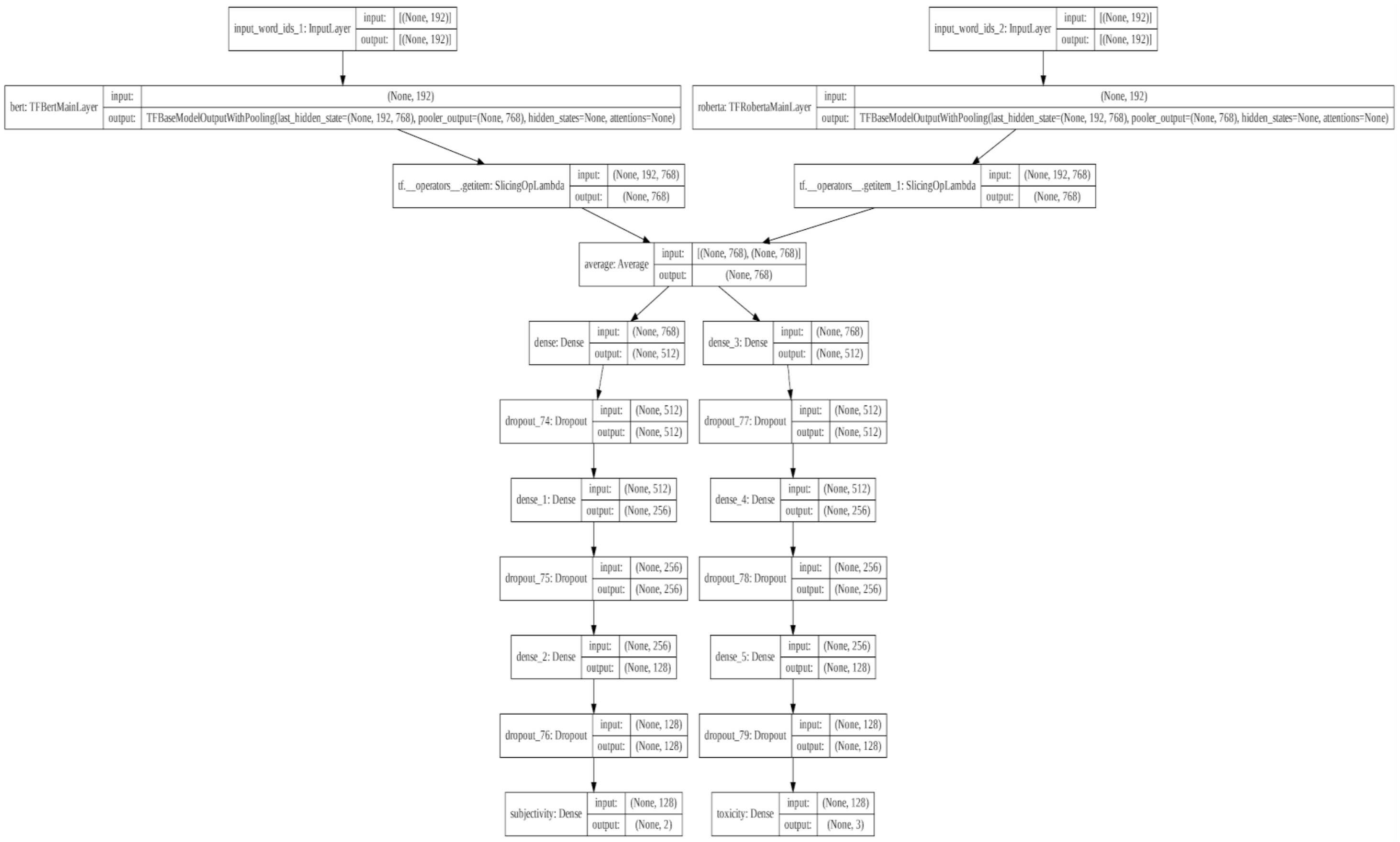


Figure 26: Multi-head Model Architecture

4.5 Performance of Classifiers

After training the two classifiers that we have defined based on our classification approach, we tested the performance of the classifiers against an unseen validation dataset from our crawled text corpus which was manually labelled and annotated by 2 of our group members. The validation dataset consists of 1,747 tweets that are manually labeled for subjectivity and toxicity classification. The validation dataset is also preprocessed using the same functions in section 4.2.1 before being fed into the classifiers for predictions.

Evaluation metrics

In order to evaluate how well the two classifiers are able to classify the text data for subjectivity classification and toxicity classification, we will be using F1 score, precision score and recall score to evaluate the ability of the classifiers to make predictions for each of the class labels.

The precision score is the number of True Positives divided by the number of True Positives and False Positives. It can be thought of as a measure of a classifier's exactness, a low precision can also indicate a large number of False Positives. The recall score is the number of True Positives divided by the number of True Positives and the number of False Negatives. It can be thought of as a measure of a classifier's completeness, a low recall indicates many False Negatives. Finally, the F1 Score is defined as 2*((precision*recall)/(precision+recall)). It is also called the F Score or the F Measure. The F1 score helps convey the balance between the precision and the recall.

Performance metrics

In addition to evaluating the ability of the classifiers to make predictions for each of the class labels, we will also be evaluating the speed of the classifiers in making predictions on records in terms of records classified per second.

4.4.1 Ensemble Classifier Results

Evaluation metrics

Subjectivity Classification			
Class	F1 Score	Precision Score	Recall Score
Neutral Class	0.87585891	0.88110599	0.87067395
Subjective Class	0.79328757	0.78549849	0.80123267

Table 4: Ensemble Classifier Subjectivity Classification Results

Toxicity Classification			
Class	F1 Score	Precision Score	Recall Score
Non-Toxic Class	0.93964583	0.93998554	0.93930636
Toxic Class	0.60903733	0.63265306	0.58712121
Severely Toxic Class	0.71559633	0.65546218	0.78787879

Table 5: Ensemble Classifier Toxicity Classification Results

Performance metrics

Speed of Classifying Records (records classified/second)
155.83

Table 6: Ensemble Classifier Classification Speed Results

4.4.2 Multi-Head Classifier Results

Evaluation metrics

Subjectivity Classification			
Class	F1 Score	Precision Score	Recall Score
Neutral Class	0.87031963	0.87271062	0.86794171
Subjective Class	0.78220859	0.77862595	0.78582435

Table 7: Multi-head Classifier Subjectivity Classification Results

Toxicity Classification			
Class	F1 Score	Precision Score	Recall Score
Non-Toxic Class	0.93016453	0.9415248	0.91907514
Toxic Class	0.55952381	0.5875	0.53409091
Severely Toxic Class	0.61176471	0.5	0.78787879

Table 8: Multi-head Classifier Toxicity Classification Results

Performance metrics

Speed of Classifying Records (records classified/second)
896.00

Table 9: Multi-head Classifier Classification Speed Results

4.4.3 Discussion of Classifiers Results

From the evaluation metrics above, we are able to observe that the Ensemble Classifier outperforms the Multi-Head Classifier for both the Subjectivity Classification and Toxicity Classification tasks with higher F1 scores, Precision scores, and Recall scores for almost all of the classes. This can be due to the fact that for the Ensemble Classifier, the Bidirectional Encoder Representations from Transformers (BERT) and Robustly Optimized BERT Pretraining Approach (RoBERTa) models are trained separately with their classification predictions only ensembled together after the classification predictions are made separately on each model which means that the models are unaffected by each other and the overall Ensemble Classifier is less overfitted and can make predictions on previously unseen data more accurately. As for the Multi-Head Classifier, the embeddings obtained by the 2 BERT and RoBERTa heads of the Classifier from the input training text data are averaged together whilst training, the weights learnt by the BERT and RoBERTa models may be affected by each other and the Multi-Head Classifier overfits more easily which causes it to be less accurate when making predictions on previously unseen data.

As for the performance of the Classifiers for the speed of the classifiers in making predictions on records in terms of records classified per second, the Multi-Head Classifier is able to classify much more records per second as compared to the Ensemble Classifier due to the fact that the prediction only has to be ran once for the Multi-Head Classifier while the prediction has to be ran twice in order to obtain the ensemble results for the 2 models of the Ensemble Classifier.

5. Submission Links

3. A YouTube link to a video presentation of up to 5 minutes

After completing the CZ4034 Information Retrieval Course Assignment, our team created a video to explain the applications and the impact of our work as well as highlight the creative parts of our work. The YouTube link to our team's video presentation is as follows:

https://www.youtube.com/watch?v=5jvYjlAXrKs&ab_channel=GabriellaBenedicta

4. A Dropbox (or Google Drive) link to a compressed (e.g., zip) le with crawled text data, queries and their results, manual classifications, automatic classification results, and any other data for Questions 3 and 5

https://drive.google.com/file/d/1mROyXwdaSKCn6VdxbOdVEE5ZmCsIpEyS/view?usp=sharing

The files in side the zipped folder are as follows:

- automatic_classification_labeled_tweets (text classified by our classification model)
- crawled_text_corpus_data (all the crawled text data from Twitter for our group's Information Retrieval System)

- manually_classified_training_data (manually annotated text data to be used for training the classification model)
- manually_classified_validation_dataset (manually annotated text data as the evaluation dataset manually by labeling 10% of the collected data)
- preprocessed_kaggle_data_for_question4 (additional preprocessed data from Kaggle to be used for training the classification model)
- queries-and-results (queries and their results)

5. A Dropbox (or Google Drive) link to a compressed (e.g., zip) file with all your source codes and libraries, with a readme file that explains how to compile and run the source codes

The Google Drive link to the compressed file with all source codes and libraries:

https://drive.google.com/file/d/1kUqVpYn8IMyhYxS7yfGbq9sgjup3aHA_/view?usp=sharing

For the readme file with images for the frontend source codes, it can be found in the Github link:

https://github.com/tkhang1999/CZ4034-Team-11/tree/master/frontend

Google Drive Link for both Submission point 4 and Submission point 5 zipped folder

In the event the Google Drive Links for the folders for Submission point 4 and Submission point 5 cannot be opened and downloaded, the Google Drive folder that contains the 2 zipped folders is as follow:

https://drive.google.com/drive/folders/1URKYEH4NYPQ7t0ONuEa4dxLeMaVPwEUk?usp=sharing