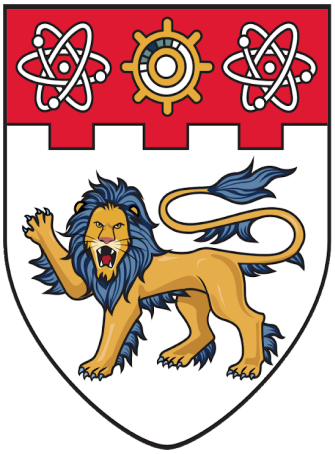

NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

CZ4034: INFORMATION RETRIVAL

Sentiment analysis on the global response to the Covid-19 vaccine using Twitter

GROUP 24

Arora Manav *(U1822077D)*
Feng Chengxuan *(U1720384G)*
Lu Cheng *(U1720359A)*
Rajasekara Pandian Akshaya Muthu *(U1721882B)*
Taneja Parthasarthi *(U1722927B)*

Table of Contents

Introduction: *Response to COVID-19 Vaccine*

Coronavirus disease (COVID-19) is an infectious disease caused by the newly discovered coronavirus. COVID-19 has a broad clinical spectrum, ranging from asymptomatic infection or mild upper respiratory tract illness to multifocal pneumonia, respiratory failure, and death.

Although it is still unknown exactly where the first case originated from, the first outbreak started in Wuhan, China in late 2019. Many early cases of COVID-19 have been attributed to people who had visited the local seafood market in Wuhan.

There have been over 131.1 million confirmed cases of COVID-19 and 2.85 million deaths worldwide. But these numbers are still growing steadily. Globally, the confirmed case fatality rate is above 5 percent, but is significantly higher in older patients. One in every 20 people with a confirmed positive COVID-19 test has died of the disease.

After a global pandemic, nearly endless lockdowns, crippling economies and a significant loss of human lives, more than a dozen vaccines now have been authorized around the globe and many more remain in development. Vaccination for this disease has reached out to 141 million people.

But quite understandably, quite some people are still concerned about the vaccination and it's safety. People can understand the urgency of authorities to push out the vaccines and, but it has also meant expediting the clinical trials. Misinformation and rumors haven't helped the cause either. In a situation like this where the world is barely recovering from the aftermath of a pandemic, sensing the global response towards vaccine becomes critically important for governments and health organisations. Meticulous data retrieval and analysis would definitely help out the people involved about the concerns the public might have and address them to promote the process of vaccination. For the relevance and the usefulness, our group have chosen to do perform sentiment analysis on the global response to the COVID-19 vaccine using Twitter data.

Crawling: *Data Retrieval*

The first step in building an efficient and reliable information retrieval system is to get a hold of high-quality relevant data, we tackled this problem by implementing a specific query-based extraction of Twitter data using a tool called snscrape. Following this, we went on to use this crawled data for indexing, querying and classification in our information retrieval system, details of which can be found further on in this document.

Crawling Method and Storage

To acquire the necessary data for our application we crawled tweets using snscrape. Snscrape is a scraper for social networking services (SNS). It scrapes things like user profiles, hashtags, or searches and returns the discovered items, e.g., the relevant posts. Currently it offers support for the following social media platforms: Facebook, Instagram, Reddit, Telegram, Twitter, VKontakte and Weibo. From the aforementioned list, it is clear that this tool is suited to our needs and is able to perform the required action, i.e., to scrape query-specific Twitter data.

Some important feature/flags of snscrape that we made use of to crawl our data are as follows:

- *Jsonl:* Outputs the data in a JSON format allowing you to access tweet information. Otherwise, you'll only receive direct links to the tweets.
- *progress:* Allows us to get updates from the CLI letting us know the progress of the scraping. It updates every 100 tweets. Does not appear to work when using Python with CLI.
- *max-results:* Puts a cap on the number of tweets scraped.
- *since:* Sets lower bound date limit on query
- *until*: Sets upper bound date limit on query

Once snscrape is set up, a sample crawling query using these flags and CLI can be written as:
snscrape --jsonl --progress --max-results Max --since Start twitter-search "Query until: End" > Filename.json

Here,

- ***Max:*** maximum number of tweets to be crawled for this query
- ***Start:*** start date or lower bound of this query
- ***Query:*** specific keyword(s) to retrieve for this query
- ***End:*** end date or upper bound of this query
- ***Filename:*** the output file to save the crawl results
-

Using this sample query format, we crawl the following data for our system:

Query	**No. of Tweets Crawled**	**Total no. of Tweets in DB**
covid vaccine	15,000	15,000
coronavirus vaccine	15,000	30,000
corona vaccine	15,000	45,000
COVID-19 vaccine	15,775	**60,775**

Table 1: *Data crawled for Information Retrieval system*

Once these individual query results were obtained all of the json files were merged and then a script was written to convert this merged json into the csv format as it was the preferred format of the team. The following script was used to convert the merged json file into a csv format.

```
import pandas as pd
with open('covid4.json', 'rb') as f:
    data = f.readlines()
data = map(lambda x: x.rstrip(), data)
data_json_str = b"[" + b','.join(data) + b"]"
data_df = pd.read_json(data_json_str)
data_df.to_csv (r'C:\twitter\covid4.csv')
```

Figure 1: *Conversion Script*

Information Type

The possible types of information that one might like to retrieve from this system are as follows:

- ➔ The total number of positive, neutral and negative responses to the vaccine
- ➔ The number of tweets containing a specific keyword
- ➔ Tweets from a specific geographic location, as our system also crawled geo-data.
- ➔ Tweets that mention specific precautionary measures and safety protocols with respect to the vaccine.
- ➔ Tweets that are talking about approval, effectiveness, access, updates and introduction of the vaccine.

Corpus Record

The following table can be referred to check the number of records, words and types in the corpus.

Total No. of Tweets	Total No. of Words		Total No. of Unique Words	
	Pre-Processing	Post-Processing	Pre-Processing	Post-Processing
60,775	1,453,183	772,001	159,051	30,237

Table 2: *Corpus Analysis*

Indexing and Querying

Indexing optimizes query efficiency by reducing the number of disks accessed required when a query is processed, allowing the queried data to be quickly located in the database. Without indexing, every row of the database must be search through linearly to find the rows that match the query conditions. With indexing, the queried column is tokenized and sorted alphabetically along with the associated 'id' (Figure 1).

The database index will also store pointers to the location of the additional information in memory. Thus, rather than searching through every single row of the database, the query looks for the specified row in the index and the rest of relevant information will located by the pointed.

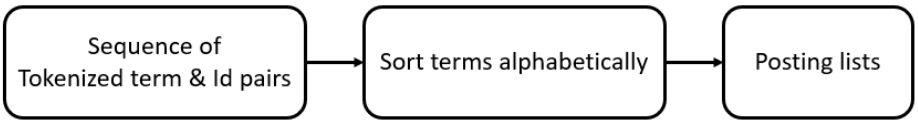

Figure 2: Indexing process

In this project, we used Apache Solr 8.8.1, a standalone enterprise search server that index documents and support advanced queries such as wildcards, joins and grouping. By querying it via HTTP GET, it returns JSON, XML, CSV or binary results.
We chose SOLR for its advanced full-text search capabilities and standard based open interfaces which allows it to be easily integrated in our web application.
After processing the crawled data, the main columns we will be querying are the processed tweets and location (Figure 2). Since the processed tweets are English, we assigned its field type as "text_en_splitting" whereby English-specific stemming and stop word removal will be applied to the field. On the other hand, the location column is assigned a "string" field type as it is a relatively small field and is not tokenized nor analyzed in any way.

```
<field name="Date" type="date" multiValued="false" indexed="true" required="true" stored="true"/>
<field name="Location" type="string" multiValued="true" indexed="true" stored="true"/>
<field name="Processed" type="text_en_splitting" multiValued="true" indexed="true" stored="true"/>
<field name="Raw" type="text_en_splitting" multiValued="true" indexed="true" stored="true"/>
<field name="id" type="string" indexed="true" required="true" stored="true"/>
```

Figure 3: *Field columns and their respective data type*

Search Engine: *A Web Interface*

The web interface was built using the Django framework. By default, the UI will query the processed tweets and location columns from Solr database and display the query result from the original tweets column. The query result was retrieved by calling Solr API using python requests library. The web interface offers two types of search, **Simple Search** (Figure 3) and **Enhanced Search** (Figure 4). Simple Search returns all relevant tweets from the query and Enhanced Search provides more statistical details of the search result using word cloud and bar-chart.

```
query_object = requests.get(query_url)
```

Figure 4: *Using Python requests library to call Solr API*

```
http://localhost:8983/solr/covid/select?q=Processed%3Acovid
```

Figure 5: *An example of Solr API*

With this web interface implemented, users can query for tweets by either content or location or both. Based on the query, it will display the corresponding ID, tweet and location. For each query, it will also display the time taken on the right-hand side in milliseconds.

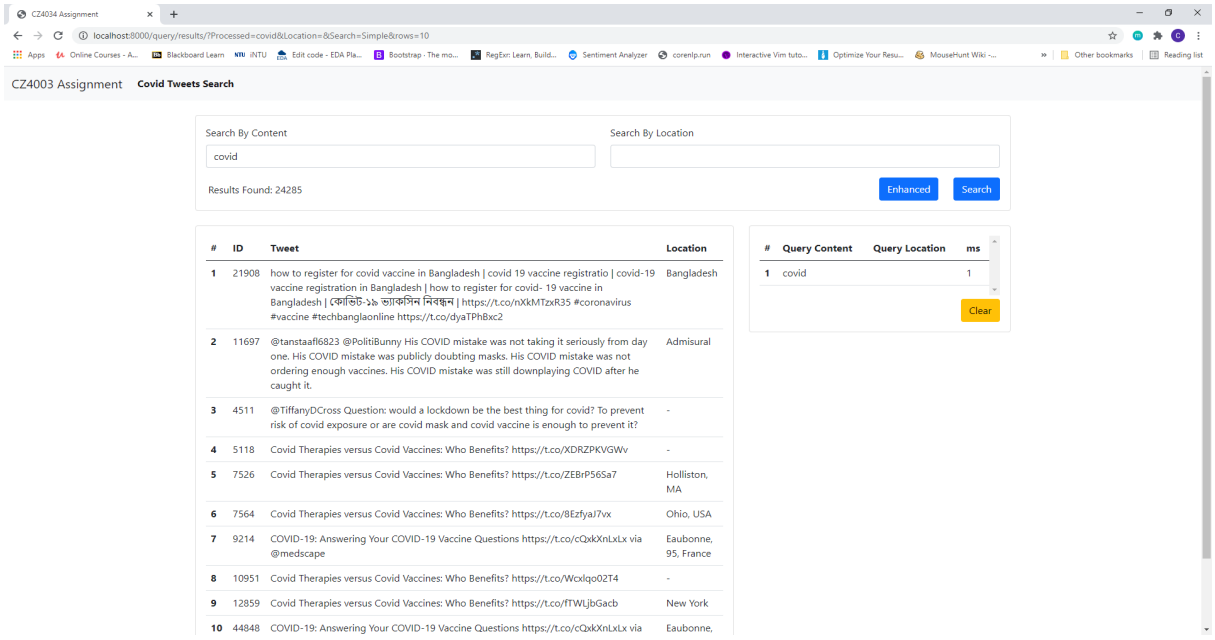

Figure 6: *Web interface for search engine*

Queries and Analysis

In this section, we explore various query types and measure the speed of querying.

Term Query

A term query is a straightforward search where a single word is used to search through the tweets.
In this search, we perform a term query to return all tweets containing the term "*hate*" in the processed field.

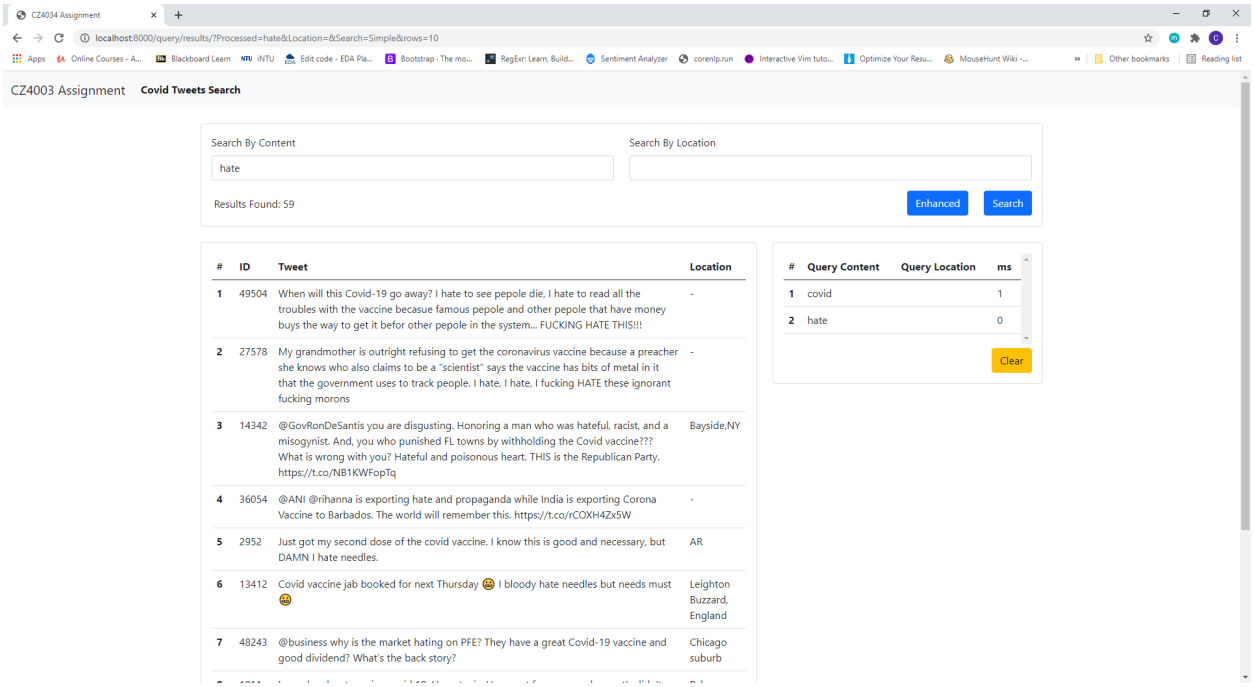

Figure 7: *Term Query result (Processed: "hate")*

The term query result shows all tweets that contain the term "*hate*" in the above screen capture.

Phase Query

A phrase query returns data that matches a sequence of terms.
In this search, we perform a phrase query to return all tweets containing the phrase “pfizer vaccine”:

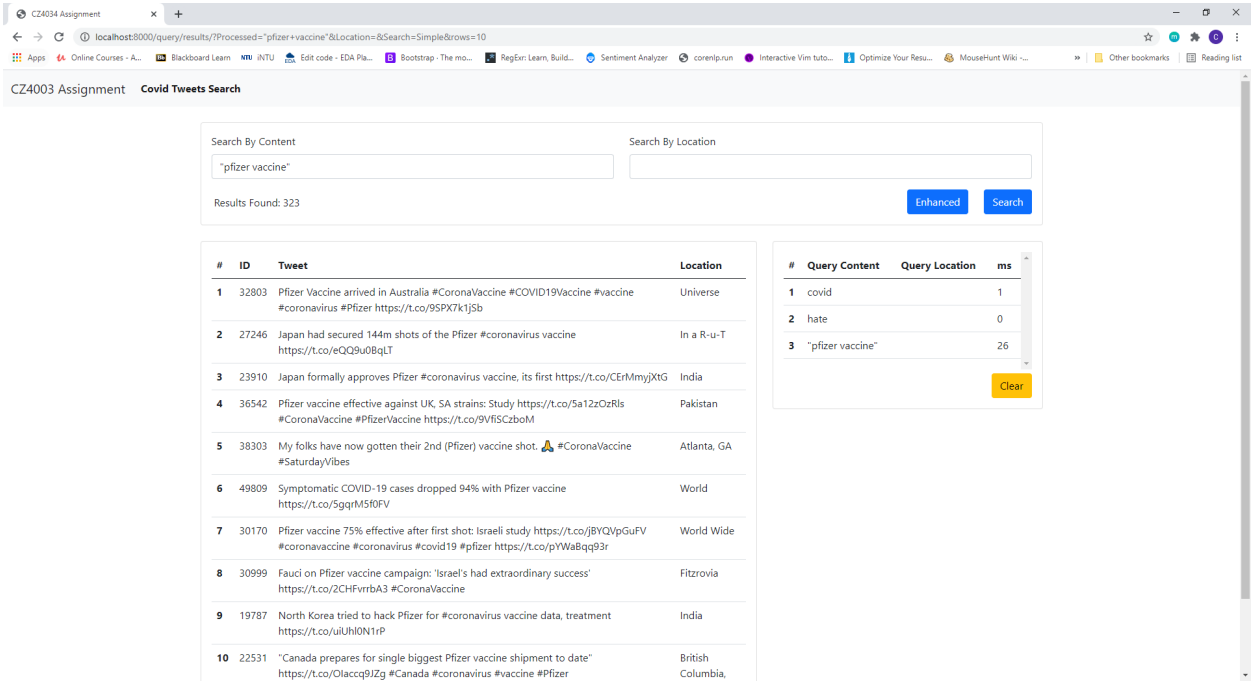

Figure 8: *Phase Query result (Processed: “pfizer vaccine”)*

The phase query results show all tweets that contain the phrase “pfizer vaccine” in the above screen capture.

Boolean Query

Boolean operators allow terms to be combined through logic operators such as AND, “+”, OR, NOT and “-”. In Solr, the OR operator is the default conjunction operator and therefore if there is no Boolean operator between two terms, the OR operator is used.
In this search, we perform a Boolean query to return all tweets that either contain the term “worry” or the term “hate”.

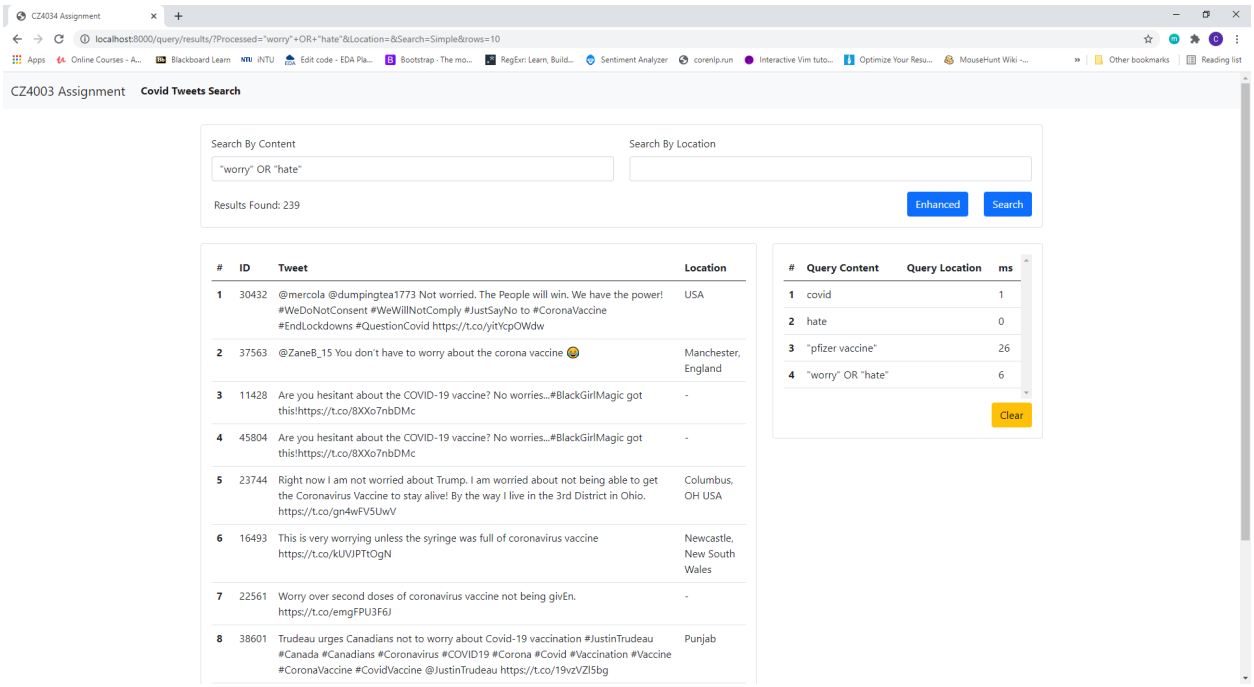

Figure 9: *Boolean Query result (Processed: “worry” OR “hate”)*

The Boolean query results show all tweets that contain the term “worry” or “hate” in the above screen capture.

Boost Query

To increase the importance of a term in a query, we can use the caret symbol ^ to boost a term and therefore controlling the relevance of a tweet. The importance of the term is controlled by the boost factor beside the caret symbol.
In this search, we perform a boost query to boost the term “angry” for tweets related to vaccine:

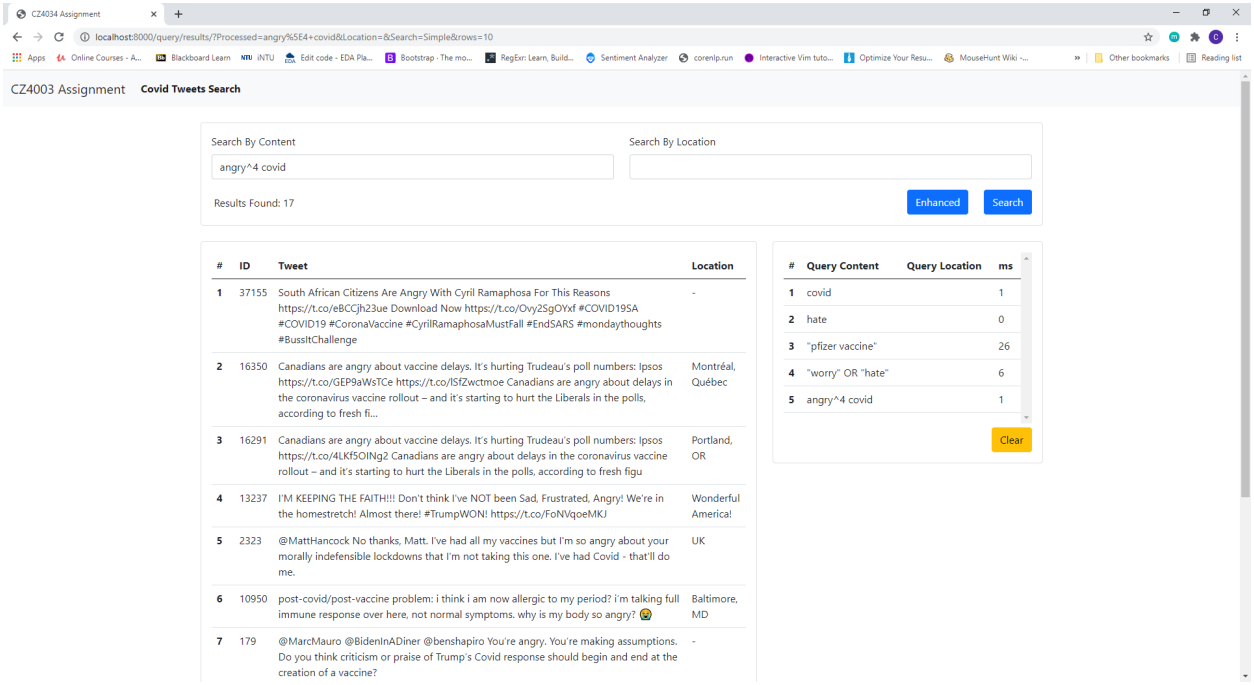

Figure 10: *Boost Query result (Processed: angry^4 covid)*

The query results show all tweets that contain the term “angry” for tweets related to vaccine in the above screen capture.

Proximity Query

A proximity query looks for terms that are within a specific distance from one another, that is the number of term movements needed to match the specified phrase.
In this search, we perform a proximity query such that there are “happy” and “vaccine” are less than 10 terms apart:

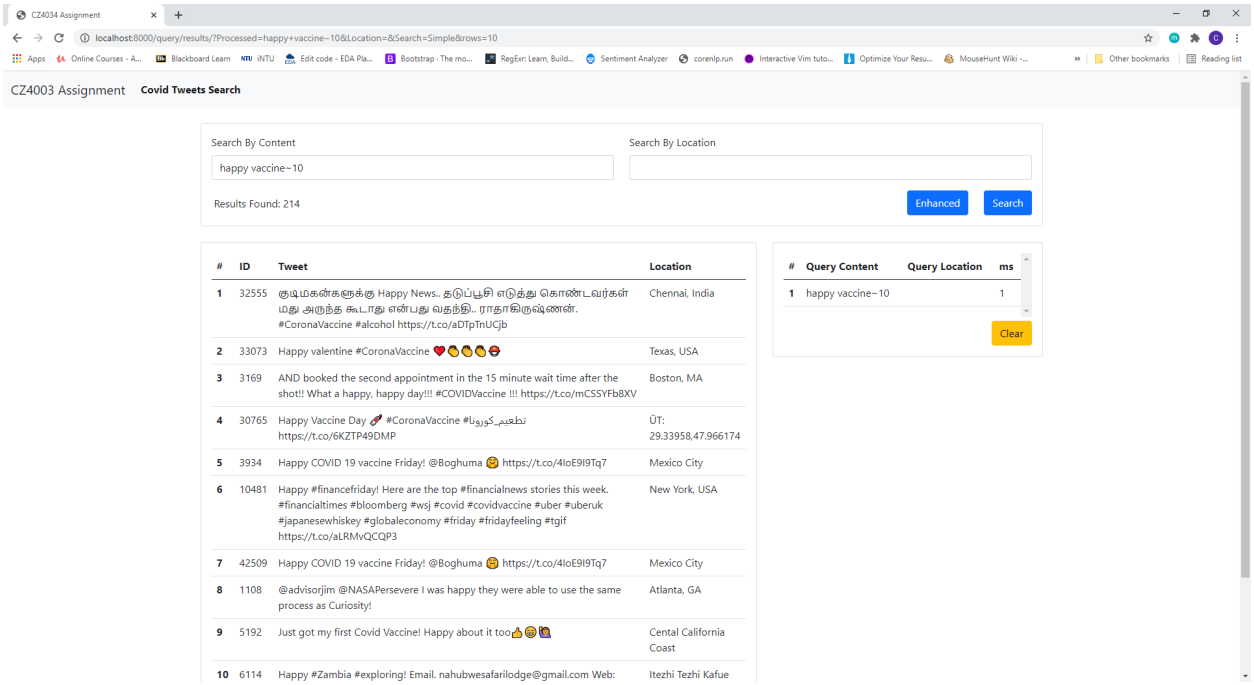

Figure 11: *Query Result of (Processed: happy vaccine~10)*

The proximity results show all tweets that contain the term “happy vaccine~10” for tweets related to vaccine in the above screen capture.

Summary:

Query	Speed of Querying (milliseconds)
"hate"	0 ms
"pfizer vaccine"	26 ms
"worry" OR "hate"	6 ms
"angry^4 covid"	1 ms
"happy vaccine~10"	1 ms

Table 3: *Speed analysis for the queries performed*

Innovation and Enhancement: *Indexing and Querying*

Due to the large number of tweets returned with every query, it is difficult for the user to see what kind of words are most commonly associated with a particular query.

Enhanced Search: *Word Cloud & Bar-Chart*

Therefore, as illustrated in Figure 11, word cloud shows a visual representation of words that give greater prominence to words that appear more frequently.

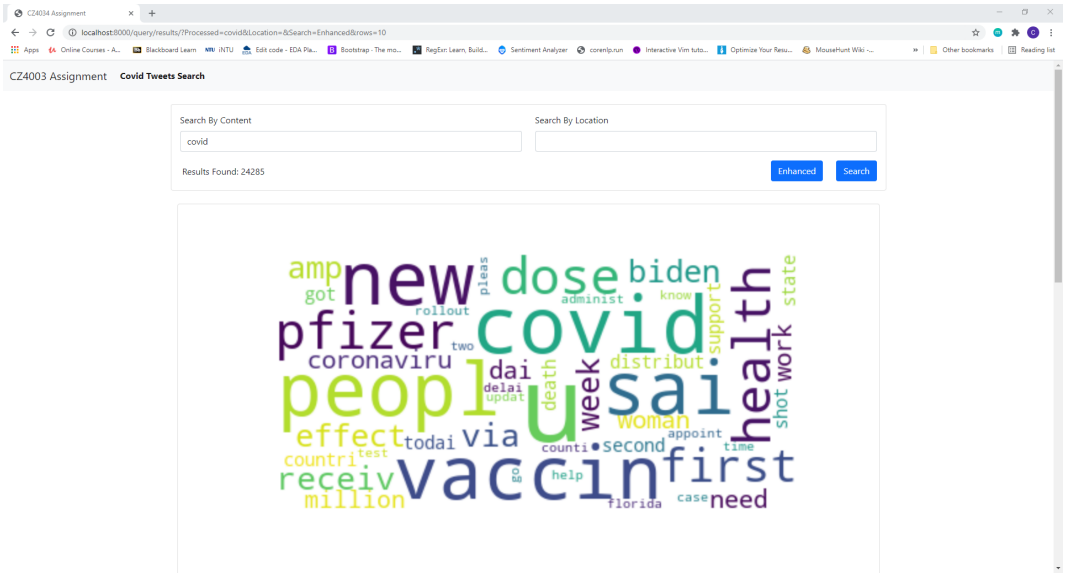

Figure 12: *Word Cloud from querying "covid"*

And as illustrated in Figure 12, bar-chart could give more statistical details about the most frequent words from the search result.

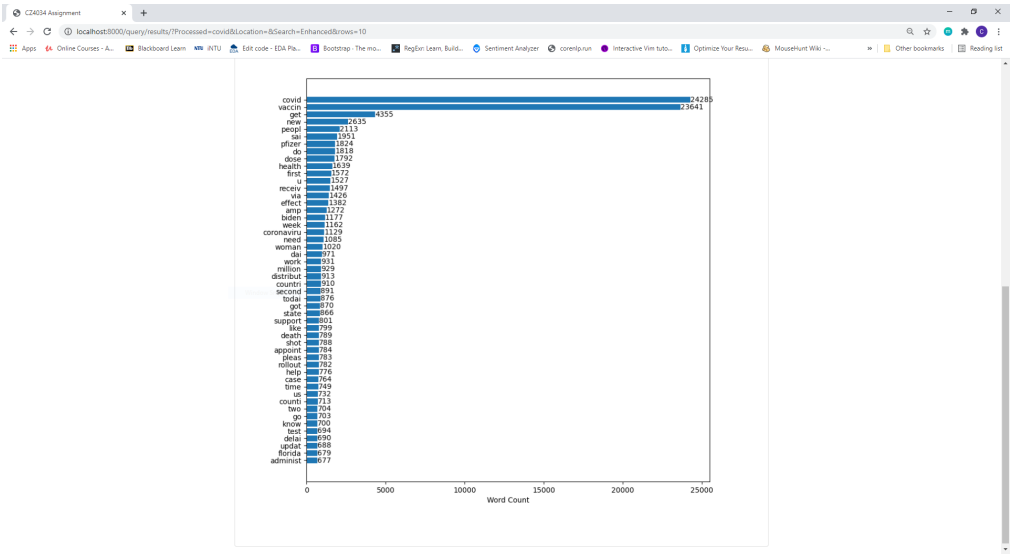

Figure 13: *Bar Chart from querying "covid"*

This is implemented using Solr's faceted search that retrieves the counts for all terms, or just the top term in any given field.

Thus, for any given query, we are able to retrieve not only the most popular terms but location as well. This will help user to understand what terms and locations are associated with the query in a single glance.

For example, by querying "vaccine" and setting the faceted field as 'Processed', Solr returns the counts of all the terms in the 'Processed' column.
By changing the faceted field to 'Location', it returns the counts of all the terms in the 'Location' column. From this, we can see that some of the most popular terms associated with vaccine is 'covid', 'new' and 'pfizer' and the most popular location is 'United States', 'Washington, DC' and 'India' (Figure 13).

Figure 14: *Faceted Search*

Enhanced Search: *Multilingual search*

In this project, we only crawled data using English but there are tweets associated with Covid-19 vaccine in other languages. To expand on the scope of this project, we can crawl the tweets in other languages such that we are able to gain a bigger perspective of the global response towards the Covid-19 vaccine.

With tweets in different languages, it presents challenges to text searching as some languages contain special accepts (French and Spanish) while others contain compound words made up of sub words (German and Dutch). To overcome these challenges, we need to detect the language being used via Tika's language detection feature supported by Solr.

By adding the UpdateRequestProcessor to the requestHandler that applies to document updates in Solrconfig.xml (Figure 15), Tika's language detector will detect the languages based on the value of the "content" field.

In this case, the content field will be mapped to either English or Spanish.

```
<processor class="org.apache.solr.update.processor.TikaLanguageIdentifierUpdateProcessorFactory">
    <lst name="defaults">
        <str name="langid.fl">Processed,Raw</str>
        <str name="langid.langField">language</str>
        <str name="langid.whitelist">en,es</str>
        <bool name="langid.map">true</bool>
        <str name="langid.map.lcmap">en:eng es:spa</str>
    </lst>
</processor>
```

Figure 15: *Configuring Tika Language Detection*

With the language identifier configured to map to sub-fields, the fields and types must be defined accordingly as well such that it will utilize the linguistic analysis (e.g. stopwords removal, tokenization) for the particular language (Figure 16).

```
<fieldType name="text_eng" class="solr.TextField" positionIncrementGap="100">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.SnowballPorterFilterFactory" language="English"/>
    <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
  </analyser>
</fieldType>
<fieldType name="text_spa" class="solr.TextField" positionIncrementGap="100">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.SnowballPorterFilterFactory" language="Spanish"/>
    <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
  </analyser>
</fieldType>
<dynamicField name="*_eng" type="text_eng" indexed="true" stored="false" multiValued="false"/>
<dynamicField name="*_spa" type="text_spa" indexed="true" stored="false" multiValued="false"/>
```

Figure 16: *Defining field types*

One way to implement multilingual search in Solr is to create a separate field for tweets crawled in a different field. On the search engine, the user can specify the language they would like to query the tweets in, and querying is simply done against a particular language's field.

Classification

Data Cleaning and Preprocessing

It is imperative to carry out text preprocessing before feeding data into the networks to ensure that the text data only has useful information and hence predictable and analyzable for our network and to allow proper feature extraction. We have carried out a few steps of preprocessing on the tweets to make them more befitting for text mining and the training of the models.

Step 1: Noise Removal and Normalisation

Since our data are tweets, we had to do some cleaning to remove unnecessary information and noise in the tweets. The dataset consisted of many tweets from news outlets and information providers and hence contained many links. We first removed any links in the tweets as links do not provide much information needed for the model training. We also removed any hashtags, usernames and punctuations in the tweets. Following these, we also carried out casefolding the tweets to all be lowercase.

Step 2: Remove Stopwords

Followed by noise removal, we removed stopwords in the data. Stopwords are a set of commonly used words in English that can be considered as noise in the text as they do not provide much information and lexical content about the text data. Removal of such words can allow the network to focus on the more important information in the text during training.

Examples of stopwords in English are “a”, “the”, “is” and “are”. We used the existing English stopwords set imported from the nltk corpus dataset. In addition to the imported stopwords set, we added new words to the set such as ‘news’, ‘new’ and ‘broadcast’ which are common in the used dataset but provide little information about the sentiment of the tweets.

Step 3: Lemmatization of Tweets

Lemmatization is carried out to reduce the various inflectional and variant forms of words to their base forms in the tweets. We used the WordNetLemmatizer imported from the nltk. Stem package. The WordNetLemmatizer() is implemented using WordNet a large and publicly available lexical database for the English language aiming to establish structured semantic relationships between words. Lemmatization allows for better context extraction during the model training.

Step 4: *Stemming of Tweets*

Stemming is carried out to reduce different word forms to their 'roots' before indexing. In this project, we used the SnowballStemmer() imported from the nltk.stem package. The SnowballStemmer algorithm is a modification of the Porter algorithm to carry out more aggressive and accurate stemming.

Step 5: *Tokenisation*

Tokenisation involves breaking down and splitting the tweets to smaller units such as words known as tokens. We used word_tokenize() imported from the nltk.tokenize package from the tokenisation of our tweets data. The tokenizer splits the tweets to tokens based on white space and punctuation.

Comparison Models

Comparison Model I: *Naive Bayes Classifier*

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable.

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering.

They require a small amount of training data to estimate the necessary parameters. Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one-dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

Comparison Model II: *K-Nearest Neighbors Classifier*

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point and predict the label from these.

The number of samples can be a user-defined constant (k-nearest neighbor learning) or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice.

Neighbors-based methods are known as non-generalizing machine learning methods, since they simply "remember" all of its training data.

Despite its simplicity, Nearest Neighbor has been successful in a large number of classification and regression problems, including handwritten digits and satellite image scenes. Being a non-parametric method, it is often successful in classification situations where the decision boundary is very irregular.

Comparison Model III: *Support Vector Machine Classifier*

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.
A support vector machine constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification, regression or other tasks. I

Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

SVMs are effective in high dimensional spaces and are still effective in cases where the number of dimensions is greater than the number of samples. It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
Although, if the number of features is much greater than the number of samples, they tend to overfit.

Comparison Model IV: *Decision Tree Classifier*

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

Some advantages of decision trees are that they are simple to understand and to interpret as trees can be visualized. They require little data preparation compared to other methods. Furthermore, the cost of using the trees is logarithmic in the number of data points used to train the tree. And it is able to handle multi-output problems.
Finally, it uses a white box model so if a given situation is observable in a model, the explanation for the condition is easily explained by Boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.

On the other hand, these classifiers tend to overfit easily and small variations in data can lead to a totally different solution. (This is mitigated using Ensemble Learning which will be explained further on in the document). Decision Trees can at times create bias if certain classes dominate the dataset.

Proposed Models

Two models are presented as the proposed models: Bidirectional LSTM (BILSTM), and fine tuning of the model, Bidirectional Encoder Representations from Transformers (BERT). The first model employs a bidirectional LSTM layer that takes word embeddings as input and culminates in a dense softmax layer. The second model encodes WordPiece embeddings and produces word vectors which are then fed to a simple dense softmax layer for linear classification.

Proposed Model I: *Bidirectional Long Short-Term Memory (Bi-LSTM) RNN*

For the first model, a bidirectional long short-term memory (Bi-LSTM) network was implemented. LSTM is a type of recurrent neural network (RNN) that excels in detecting and learning long term dependencies compared to other RNNs. LSTM performs well for sequential data such as tweets as it is able to store previous timestamp information and learn sequential information. Bi-LSTM is implemented by putting 2 independent LSTMs together and allows the network to learn both backward and forward information which improves the performance of the network especially when using sequential data such as tweets.

Model Overview

- Embedding Layer
- Bidirectional LSTM Layer
- Output Softmax Layer

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 182, 50)           1228350
_________________________________________________________________
dropout (Dropout)            (None, 182, 50)           0
_________________________________________________________________
bidirectional (Bidirectional (None, 128)               58880
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 3)                 387
=================================================================
Total params: 1,287,617
Trainable params: 59,267
Non-trainable params: 1,228,350
_________________________________________________________________
```

Figure 17: *Model Summary of Bi-LSTM Model*

Model Architecture

Embedding Layer

The embedding layer was implemented using the GLOVE vector embeddings for the embedding of the input tweets. The embeddings are pretrained on twitter data and the 50-dimensional embeddings was utilised for this project. Since the word embeddings are pretrained on twitter data, this results in domain specific embeddings which allow for better learning of weights during the training of the model. The embedding layer is initiated with the weights from the GLOVE embeddings and the trainable variable is set to be false as we do not want the layer to be updating the implemented weights as training progresses.

Bi-LSTM Layer

The Bi-LSTM layer is implemented with 64 LSTM calls. The heavy lifting of learning word level features is done by this layer. Some sentence level features are also learnt in this layer. The layer is implemented with a dropout rate of 0.3 to prevent overfitting and l2 regularisation of 10-5. Return_sequences is set to True to obtain each LSTM cell's output. [add more info]

Output Softmax Layer

The output of the Bi-LSTM layer is passed to the softmax layer of 3 units as there are 3 classes to be classified, positive, neutral and negative.

Model Loss and Optimisation

The Bi-LSTM model uses the keras built-in CategoricalCrossentropy loss function which computes the cross-entropy loss between the ground truth labels and model predictions. Since there are more than 2 label classes and the labels are fed to the model as one hot representation, the categorical cross entropy loss function is the most suitable for this model.

The Bi-BILSTM model uses the Adadelta optimiser which implements the Adadelta algorithm. Adadelta is a stochastic gradient descent method which is an extension of Adagrad optimization. The optimiser adapts and modifies the learning rate according to a moving window of gradient updates and which allows Adadelta to continue learning even with the continual update of the learning rate as training progresses. We have set the initial learning rate as 1.0.

Proposed Model II: *Bidirectional Encoder Representations from Transformers (BERT)*

The second model, BERT is a Transformer-based machine learning technique for natural language processing (NLP) pre-training developed by Google. BERT is the state-of-the-art model that outperforms other methods as it is an unsupervised deeply bidirectional system for training. BERT utilises

Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its vanilla form, Transformer includes two separate mechanisms — an encoder that reads the text input and a decoder that produces a prediction for the task. As opposed to directional models, which read the text input sequentially (left-to-right or right-to-left), the Transformer encoder reads the entire sequence of words at once.
Therefore, it is considered bidirectional, and this characteristic allows the model to learn the context of a word based on all of its surroundings (left and right of the word), making it consider some global information and outperform other models.

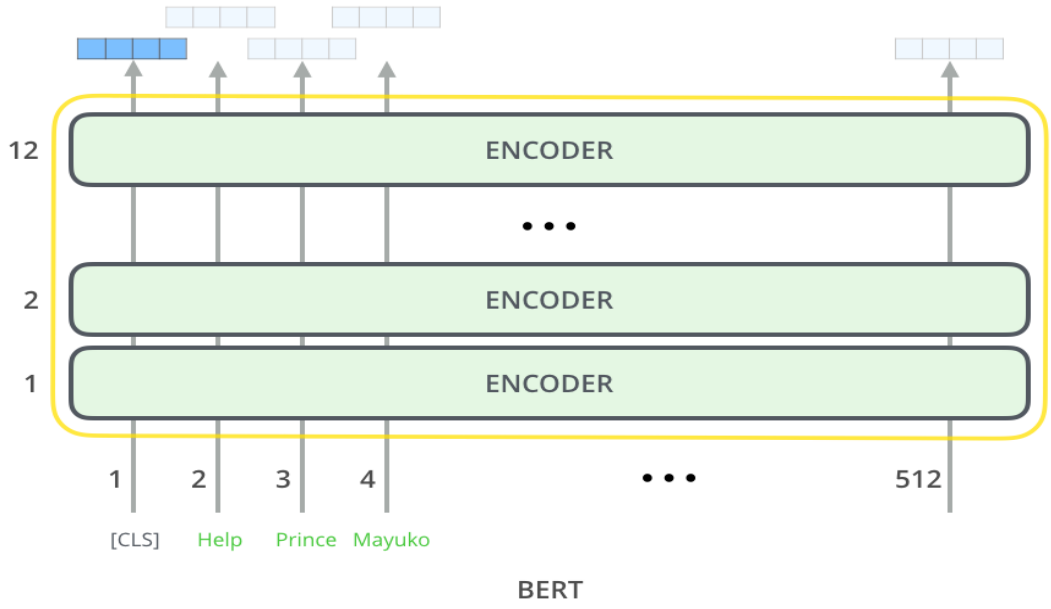

Figure 18: *Model Summary of Bi-LSTM Model*

For implementing BERT, a library named Transformers by huggingface.co was used. The library consists of state-of-the-art pre trained models. We used the **bert-base-uncased** model from the **TFBertForSequenceClassification** module for our model. The model only uses an encoder, rather than the usual encoder decoder structure. The module is designed for fine tuning the actual BERT model for our tweet sentiment classification problem. The bert-base-uncased is a BERT model trained on lower case English data and pretrained on a large corpus of English data.

Model Overview

```
Model: "tf_bert_for_sequence_classification"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bert (TFBertMainLayer)       multiple                  109482240
_________________________________________________________________
dropout_37 (Dropout)         multiple                  0
_________________________________________________________________
classifier (Dense)           multiple                  3845
=================================================================
Total params: 109,486,085
Trainable params: 109,486,085
Non-trainable params: 0
_________________________________________________________________
```

Figure 19: *Model Summary of Bi-LSTM Model*

Model Architecture

BERT Tokenizer

The BERT tokenizer is based on the WordPiece tokenizer. WordPiece is a subword tokenization algorithm which first initializes the vocabulary to include every character present in the training data and progressively learns a given number of merge rules based on the most frequent combinations of symbols in the vocabulary.

BERT

The BERT model we are using bert-base-uncased, is an encoder stack of transformer architecture. The BERT base network has 12 layers in the encoder stack with a large feedforward network. The network has 768 hidden units, 12 attention heads and approximately 110M parameters. The network accepts classification tokens followed by a sequence of words as input. The results pass through the feedforward network and each encoder applies self-attention, before passing to the next encoder. The output from the final 12th encoder layer is passed to a classifier layer.

Model Loss and Optimisation

The BERT model uses the keras built-in **SparseCategoricalCrossentropy** loss function which also computes the cross-entropy loss between ground truth labels and model predictions. Contrary to the Bi-LSTM model, this model uses integers to represent the label data and hence making this loss function the appropriate loss function for the BERT model.

The BERT model uses the LAMB optimiser which is a Layer-wise adaptive optimizer which uses momentum. It extends from the LARS optimiser and AdamW optimiser and adds a per weight normalisation with respect to the square root of the second moment to compute the update of the learning rate and updates the learning rate using a more accurate layer-wise clipped trust ratio. Therefore, LAMB allows training of the model with large mini batches with no compromise in the accuracy of the model. We have set the initial learning rate as 2e-5 and epsilon as 1e-08.

Data Annotation: *Manual Labeling*

Since the crawled data was unlabeled, all data had to be hand-labelled by our team for training and testing of the models. The data was split between the 5 team members and hand-labelled over a period of one week. Since individual human judgements for the labelling of the data might be inconsistent and there are inconsistencies between the raters, we carried out inter-judge agreement and cross-checking to ensure some consistency in the sentiment labelling of the data.

Every rater's data was cross-checked by another rater and disagreements were noted and discussed by the team. The labels were changed according to the consensus of the team and the inter-judge agreement scores were calculated and the team made sure that the probability that raters agreed was greater than 80% for all raters. The final average proportion of the times the judges agreed is 0.99021.

Rater 1 relevance					
Rater 2 relevance		**Positive**	**Neutral**	**Negative**	**Total**
	Positive	19321	239	0	19,560
	Neutral	145	23982	166	24,293
	Negative	0	178	13423	13,601
	Total	19,466	24,399	13,589	**57,454**

Table 4: *Rater Relevance*

Evaluation Overview

Setup

Train-Test Split

- Split arrays or matrices into random train and test subsets
- Our chosen split was 90-10, i.e., train - 90% and test 10%

Vectorization

Count Vectorizer

Converts a collection of text documents to a matrix of token counts and produces a sparse representation of the counts. It has a parameter called ngam_range which sets the lower and upper boundary of the range of n-values for different word n-grams or char n-grams to be extracted. All values of n such that *min_n <= n <= max_n* will be used. For our use case, this range was set to (1,2), i.e., we used unigrams and bigrams.

TF-IDF Vectorizer

Converts a collection of raw documents to a matrix of TF-IDF features. It is equivalent to CountVectorizer followed by TfidfTransformer. In a large text corpus, some words will be very present hence carrying very little meaningful information about the actual contents of the document.
If we were to feed the direct count data directly to a classifier those very frequent terms would shadow the frequencies of rarer yet more interesting terms. In order to re-weight the count features into floating point values suitable for usage by a classifier we use the tf–idf transform. Here, tf means term-frequency while tf–idf means term-frequency times inverse document-frequency.

While the tf–idf normalization is often very useful, there might be cases where the binary occurrence markers might offer better features. Which is why we implement both tf-idf and Count Vectorizers. In particular, some classifiers such as Naive Bayes explicitly model discrete Boolean random variables and tend to perform better with count vectorization. Also, very short texts are likely to have noisy tf–idf values while the binary occurrence info is more stable.

Metrics

Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. High precision relates to the low false positive rate.

Precision = True Positives / True Positives + False Positives

Recall

Recall is the ratio of correctly predicted positive observations to all observations in actual class.

Recall = True Positives / True Positives + False Negatives

F1 Score

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. F1 is usually more useful than the standard accuracy, especially if one has an uneven class distribution.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Evaluation of Comparison Models

Evaluation of Comparison Model I: Naive Bayes Classifier

We ran experiments using the Naive Bayes Classifier using both the count vectorizer and tf-idf vectorizer. Using count vectorizer we achieved a F1 score, precision score and recall score of 0.824, 0.833 and 0.824 respectively. Using tf-idf vectorizer, we achieved a F1 score, precision score and recall score of 0.800, 0.812 and 0.800 respectively.

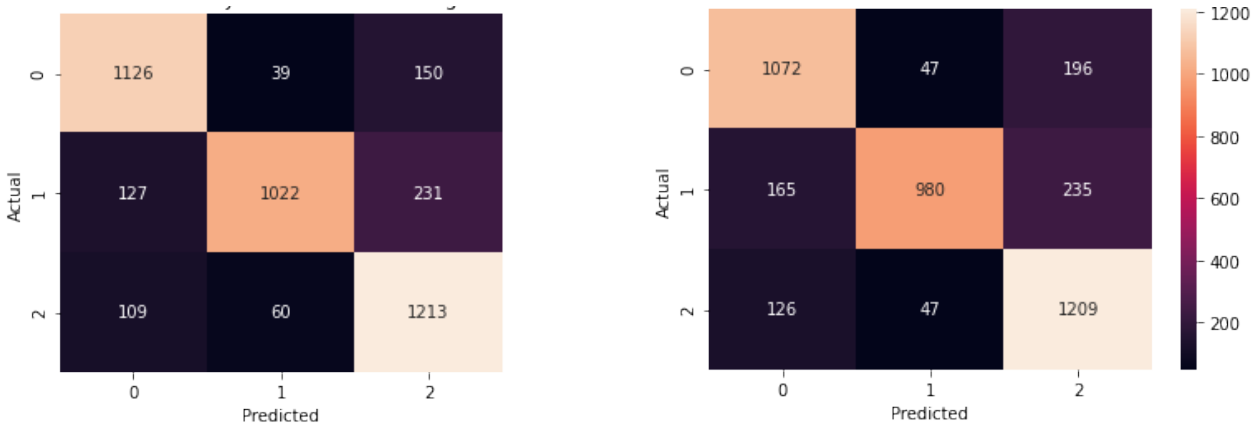

Figure 20: *Confusion Matrix using Countvectorizer (left) and tf-idf vectorizer(right)*

Evaluation of Comparison Model II: *K-Nearest Neighbours Classifier*

When using KNN classifier with count vectorizer we achieved the F1 score, precision score and recall score of 0.788, 0.735 and 0.735 respectively. Using tf-idf vectorizer, we achieved the F1 score, precision score and recall score of 0.790, 0.757 and 0.757 respectively.

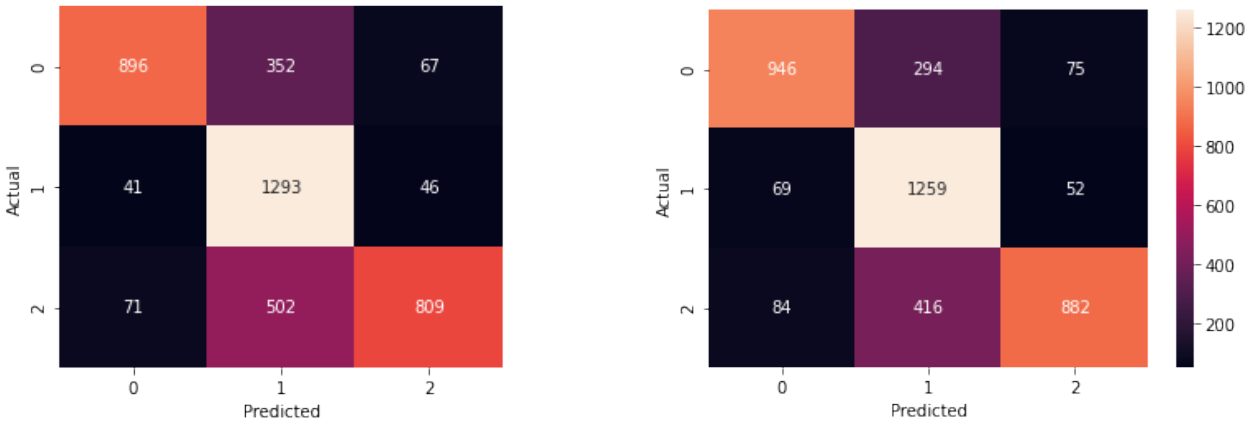

Figure 21: *Confusion Matrix using Countvectorizer (left) and tf-idf vectorizer(right)*

Evaluation of Comparison Model III: *Support Vector Machine Classifier*

Using SVM Classifier with count vectorizer with achieved a F1 score, precision score and recall score of 0.944 for all. Using tf-idf vectorizer, we achieved a F1 score, precision score and recall score of 0.953 for all. Among comparison models, SVM performed the best.

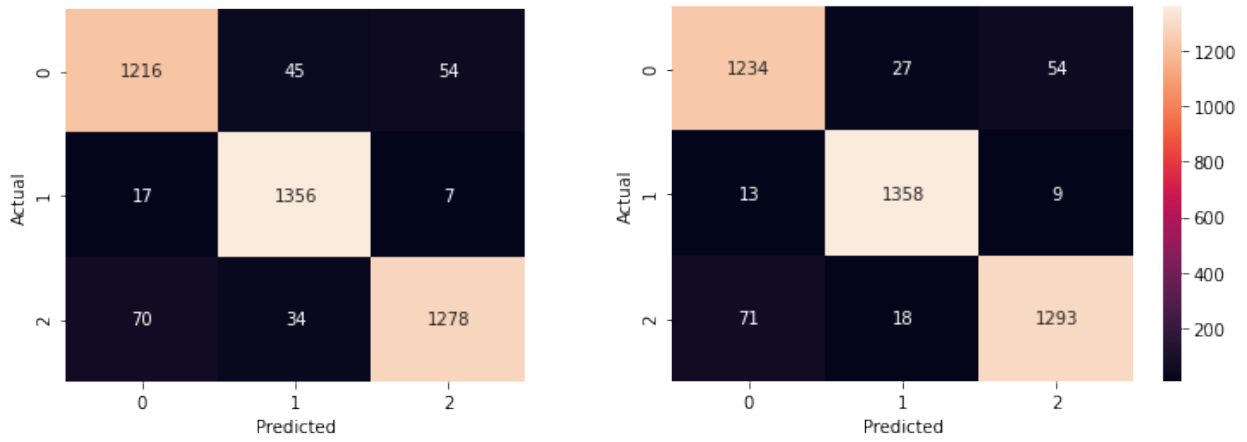

Figure 22: *Confusion Matrix using Countvectorizer (left) and tf-idf vectorizer(right)*

Evaluation of Comparison Model IV: *Decision Trees Classifier*

Using SVM Classifier with count vectorizer with achieved a F1 score, precision score and recall score of 0.801, 0.798 and 0.797 respectively. Using tf-idf vectorizer, we achieved a F1 score, precision score and recall score of 0.790 for all.

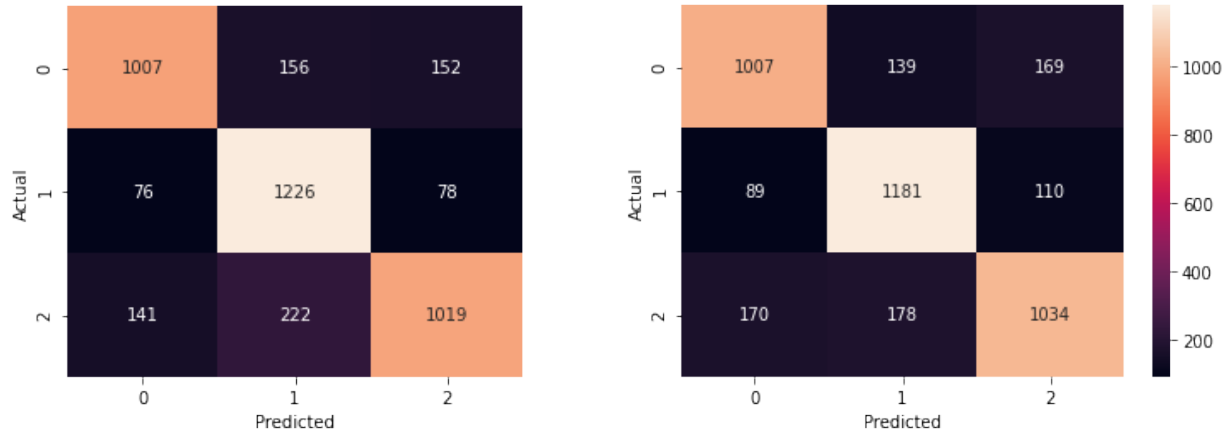

Figure 23: *Confusion Matrix using Countvectorizer (left) and tf-idf vectorizer(right)*

Evaluation of Proposed Models

Evaluation of Proposed Model I: *Bi-LSTM Model*

After training the Bi-LSTM model for 250 epochs, the following accuracy and loss plots are derived. The validation accuracy value of the model converges to approximately 0.882 and the validation loss converges to about 0.325.

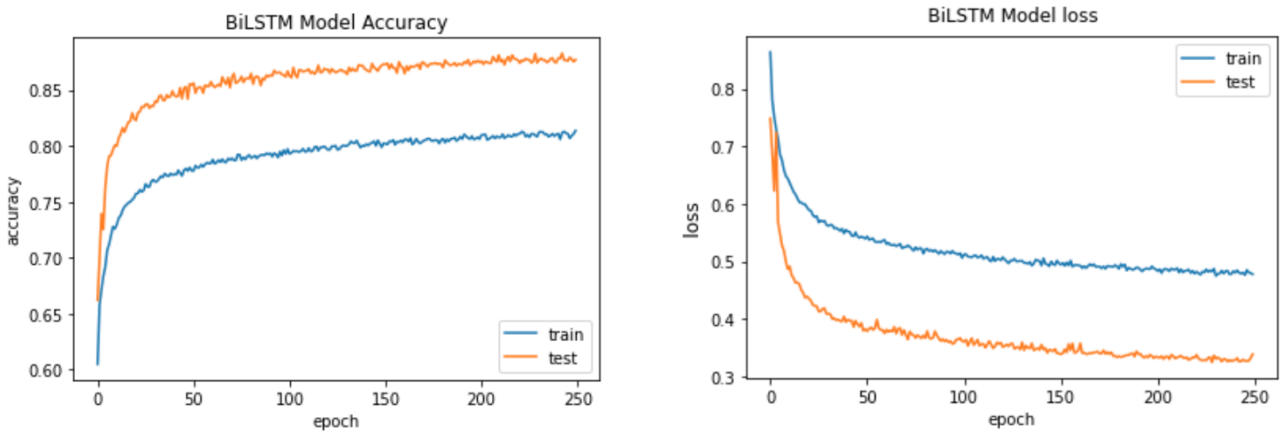

Figure 24: *Model training and validation accuracies and loss plots*

Using the trained model to predict sentiments for the testing data, we achieved a precision score, recall score and F1 score of 0.878, 0.877, 0.876 respectively. The following confusion matrix and classification report is achieved.

	precision	recall	f1-score	support
positive	0.86	0.86	0.86	1350
neutral	0.88	0.94	0.91	1392
negative	0.90	0.82	0.86	1335
accuracy			0.88	4077
macro avg	0.88	0.88	0.88	4077
weighted avg	0.88	0.88	0.88	4077

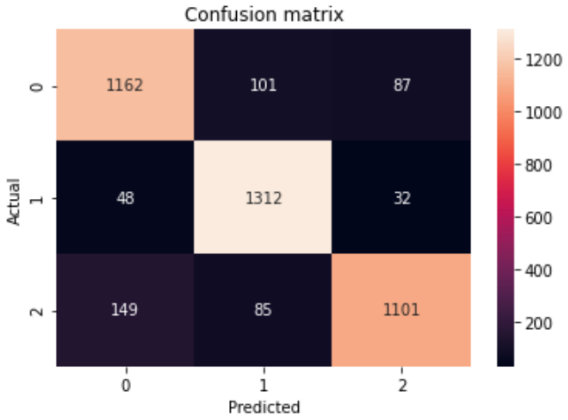

Figure 25: *Classification Report using Bi-LSTM; Confusion Matrix*

Evaluation of Proposed Model II: *BERT Model*

After training the BERT model for 5 epochs, the following accuracy and loss plots are derived. The validation accuracy value of the model converges to 0.9536 and the validation loss converges to 0.1392.

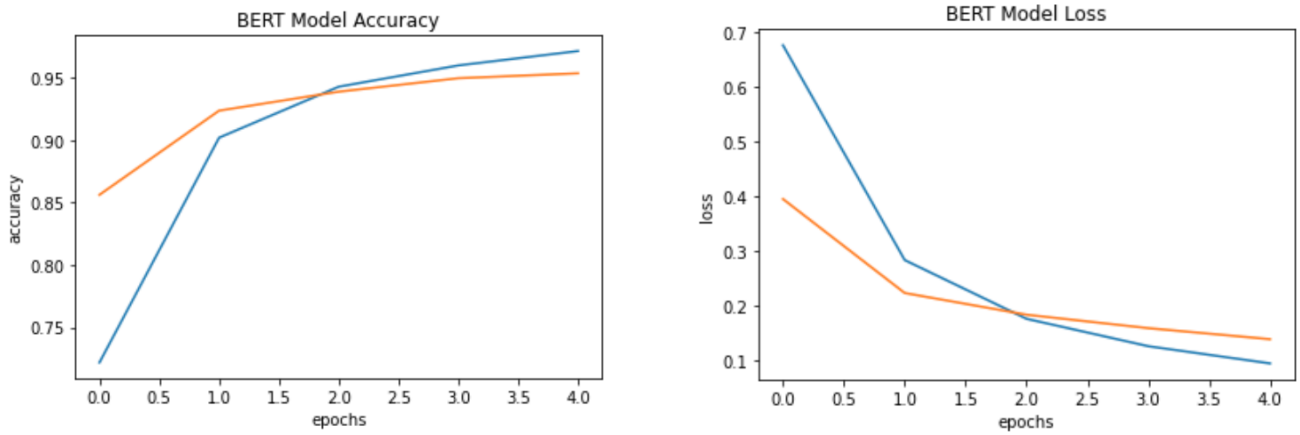

Figure 26: *Model training and validation accuracies and loss plots*

Using the trained model to predict sentiments for the testing data, we achieved a precision score, recall score and F1 score of 0.952. The following confusion matrix and classification report is achieved.

	precision	recall	f1-score	support
positive	0.96	0.93	0.94	1403
neutral	0.96	0.98	0.97	1308
negative	0.94	0.95	0.95	1366
accuracy			0.95	4077
macro avg	0.95	0.95	0.95	4077
weighted avg	0.95	0.95	0.95	4077

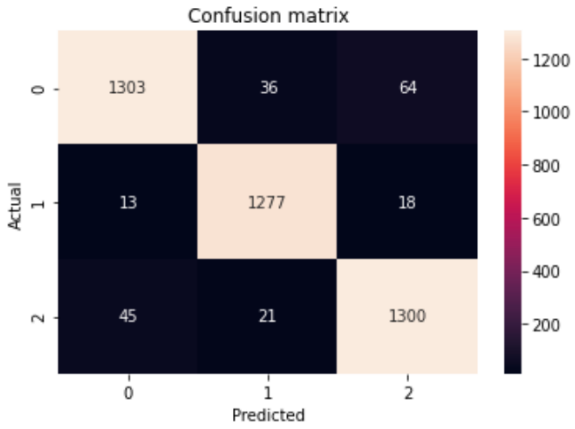

Figure 27: *Classification Report using BERT; Confusion Matrix*

Discussion of Results

Results: *Comparison Models*

Model	Count Vectorizer			Tf-idf Vectorizer		
	Precision	**Recall**	**F1 score**	**Precision**	**Recall**	**F1 score**
Naive Bayes	0.824	0.833	0.824	0.812	0.800	0.800
KNN	0.788	0.735	0.735	0.790	0.757	0.757
SVM	**0.944**	**0.944**	**0.944**	**0.953**	**0.953**	**0.953**
Decision Tree	0.801	0.798	0.797	0.790	0.790	0.790

Table 5: *Results of Comparison Models*

Results: *Proposed Models*

Label	Bi-LSTM			BERT		
	Precision	**Recall**	**F1 score**	**Precision**	**Recall**	**F1 score**
Positive	0.86	0.86	0.86	0.96	0.93	0.94
Neutral	0.88	0.94	0.91	0.96	0.98	0.97
Negative	0.90	0.82	0.86	0.94	0.95	0.95
Weighted Average	**0.876**	**0.878**	**0.877**	**0.952**	**0.952**	**0.952**

Table 6: *Results of Proposed Models*

Looking at the results, we can see that the proposed Bi-LSTM and fine-tuned BERT models perform better than most of the comparison models namely, Naive Bayes, KNN and Decision Tree for all three performance metrics.

However, SVM classifier performs better than the Bi-LSTM and comparably to the BERT model. This shows the effectiveness of our proposed models as it is able to perform considerably better than most of the comparison models.

The weighted average scores are calculated by Observing the performance of the proposed models, both models are most successful in labelling neutral tweets compared to positive and negative tweets.

Bi-LSTM achieves a marginally higher recall score than F1 and precision scores showing that the model is able to correctly predict the respective tweet labels relative to all labels. The BERT model performs similarly for all performance metrics.

Visualizing Classified Data

WordCloud

A word cloud is a collection, or cluster, of words depicted in different sizes. It is a tool used to highlight popular words and phrases based on frequency and relevance. The larger the word in the visual the more common the word was in the corpus.

As seen in Figure 26, a WordCloud with positive sentiment type reveals words like “pfizer”, “safe”, “good”, “biden” and “thanks” which resonates the positivity of the public to the vaccine.

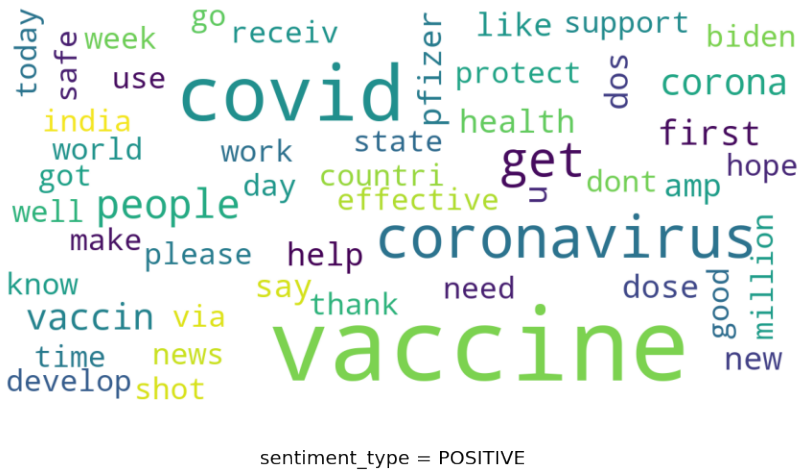

Figure 28: *Positive WordCloud*

As seen in Figure 27, a WordCloud with negative sentiment type spits out words like “death”, “poor”, “risk”, “delay” and “storm” which amplifies the sincere concerns of the public.

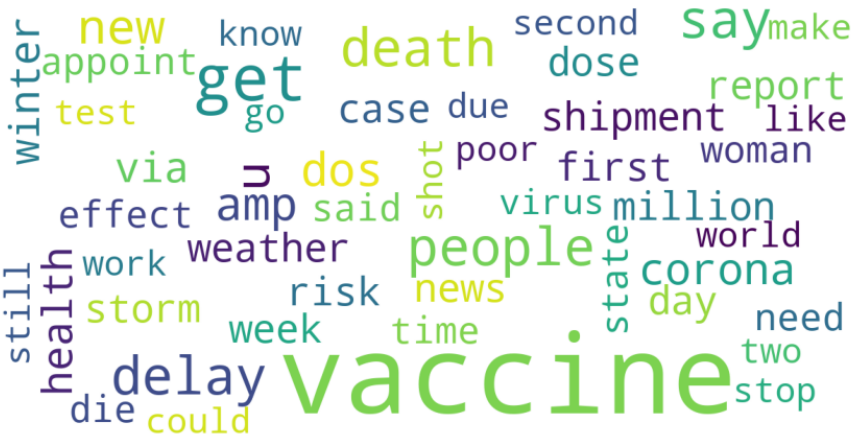

Figure 29: *Negative WordCloud*

As seen in Figure 27, a WordCloud with neutral sentiment type gives words like “million”, “first”, “update”, “govern” and “given” which doesn’t necessarily imply much meaning for conclusive sentiment analysis.

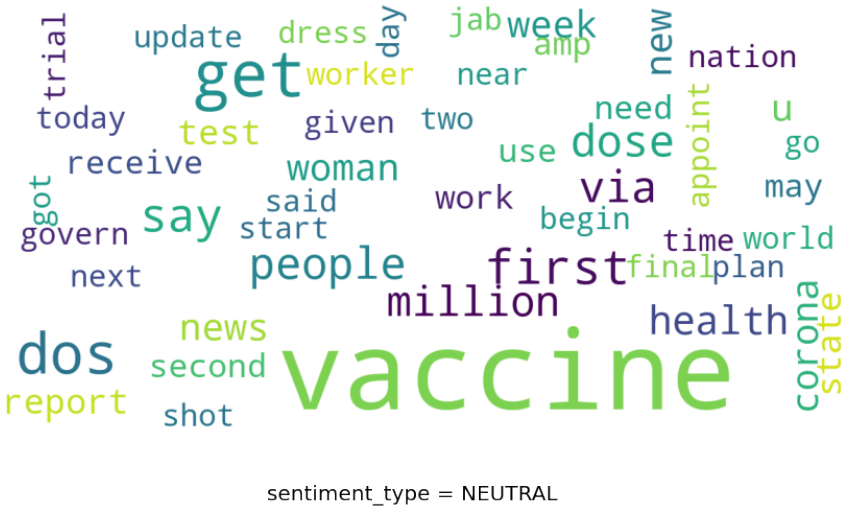

Figure 30: *Neutral WordCloud*

This type of visualization can assist evaluators with exploratory textual analysis by identifying words that frequently appear for conclusive derivations. Word clouds are attractive and can be created quickly, but they can have their own shortcomings when used as a tool for identifying actionable insights from survey data.

Topic Modelling

Topic Modelling is a type of statistical modeling for discovering the abstract “topics” that occur in a collection of documents. Latent Dirichlet Allocation (LDA) is an example of topic model and is used to classify text in a document to a particular topic. It builds a topic per document model and words per topic model, modelled as Dirichlet distributions.

For this project, we use LDA to group features that tend to co-occur in the data of same sentiment type. LDA is a probabilistic distribution algorithm which uses Gibbs sampling to assign topics to documents. In LDA a topic is a probabilistic distribution over words and each document is modeled as a mixture of topics. This means that each tweet can be associated to different topics and that topics are associated to different words with a certain probability.

To prepare good segregation topics, the data was preprocessed into tokens and lemmatization and stop words removal were then performed. Words that have fewer than 4 characters are removed.

A dictionary was then created from the data and converted to a bag-of-words corpus. The pyLDAvis library was used to analyze and discover useful insights based on the visualization created. It is a library which extracts information from a topic model and creates a visualization where users can interactively explore the model.

Interpretation:

- The red bars represent the frequency of each word given a topic.
- The blue bars represent the overall frequency of the word in the
- The area of the circle represents the topic prevalence.
- The distance between topics is the approximation of the semantic relationship between the topics. When two topics are close to each other, they are semantically related.
- By decreasing the relevance metric (lambda) on the right corner, we put more weight to the frequency of the topic as compared to the overall frequency of the word. The lower the lambda value, the easier it is to what the topic is about.
- By setting the lambda value to 0, we can see words that are exclusive to the topic
- By hovering over a specific word, we can observe where the word is used in other topics. It can be used to see how words are applied in different contexts.

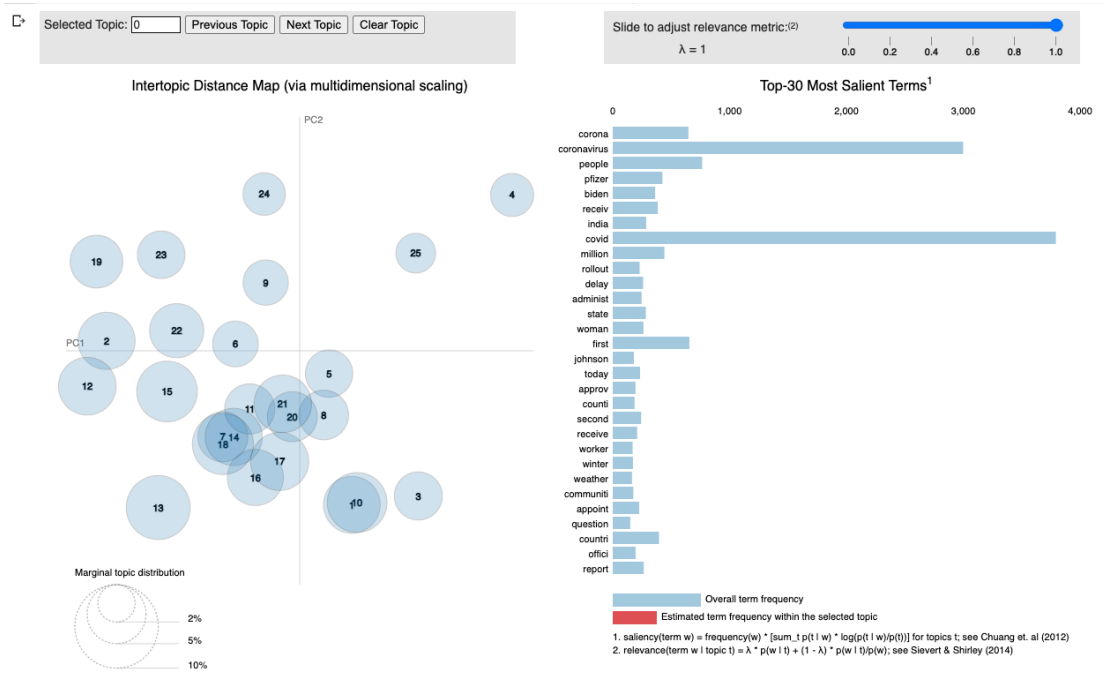

Figure 31: *LDA Visualisation*

Insights:

From the processed tweets, we can observe a high usage of the term “pfizer”. By hovering over the term, “pfizer”, we can observe that there are a few topics related to it.

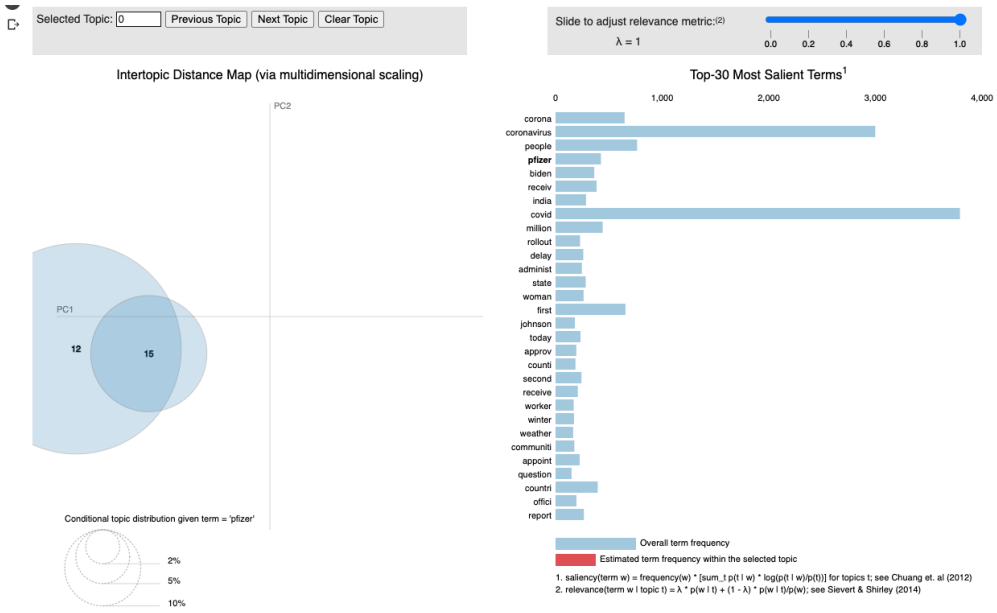

Figure 32: *Topics related to term “pfizer”*

By selecting topic-12, we could clearly see high usage of terms “BioNTech” and “Moderna” which obviously refers to the pharmaceutical giants who developed vaccines and underwent “trial”. Country names like “Japan” and “Korea” also pop up significantly due to their governments’ approval of “pfizer” vaccine after the “testing” was “satisfactory” at the time of crawling.

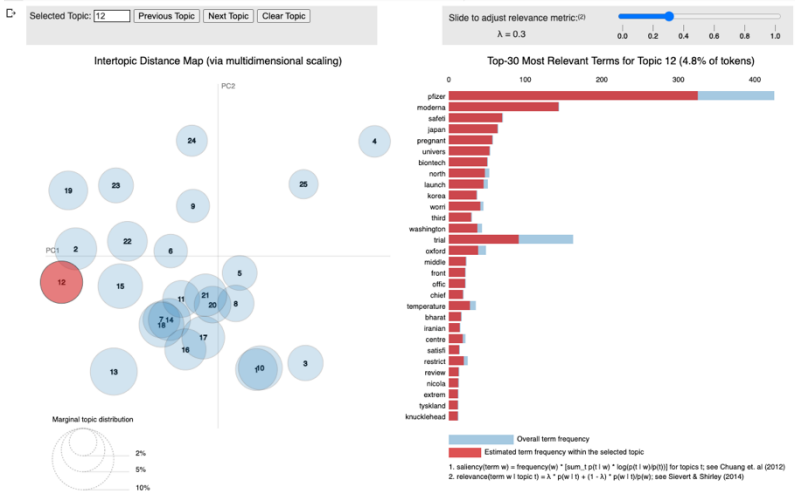

Figure 33: *Topic 12*

As seen in Figure 30, “pfizer” does have topic-12 as the major contributor. But since topic-15 is quite similar to topic-12, it calls for an inspection.

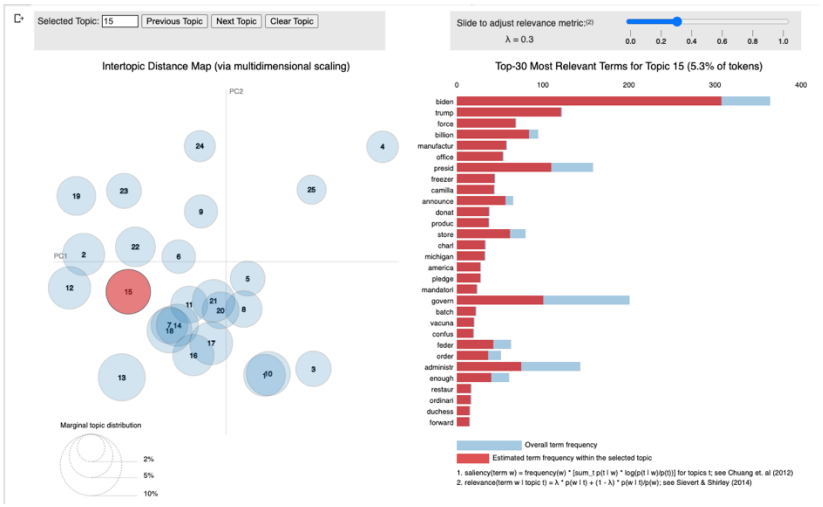

Figure 34: *Topic 15*

We can see the term list being dominated by “biden” and “trump” which resonates the debate of vaccination efforts under the two “offices”, i.e., the Biden “administration” with the Trump “administration”. It also refers to the “america” first program being continued by the “government” across “presidencies”

Among the tweets from the people, there was a strong concern regarding the “delay” of the vaccine. By hovering over the term, “delay”, we can observe that there are a substantial number of topics relating to it.

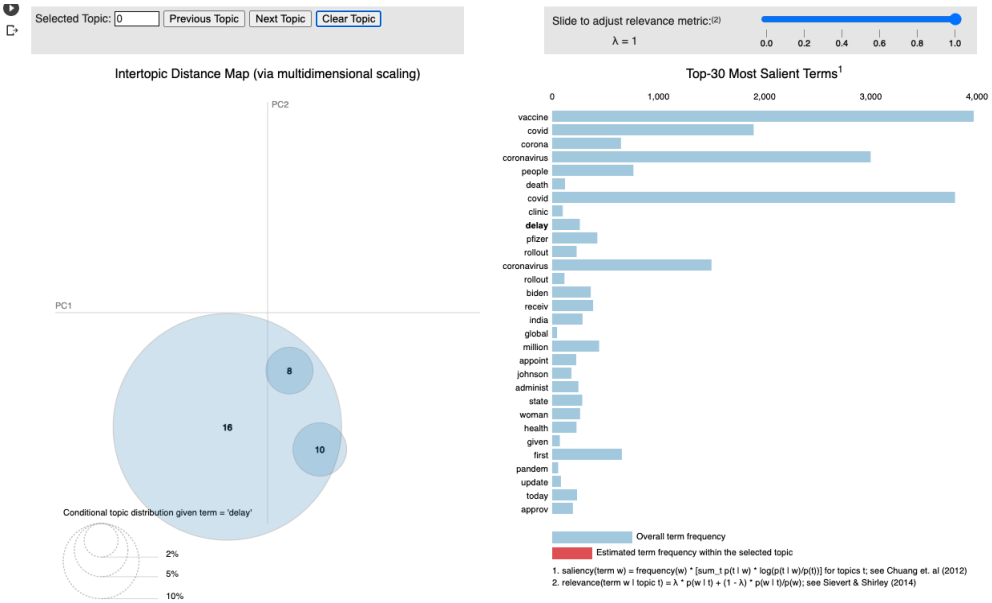

Figure 35: *Topics related to term “delay”*

By selecting topic-16, we could clearly see high usage of terms like “winter” and “weather” that clearly referred to the news story of massive winter storms across the Midwest and Texas that delayed the delivery of 6 million Covid-19 vaccine doses affecting every state in the U.S. this year.

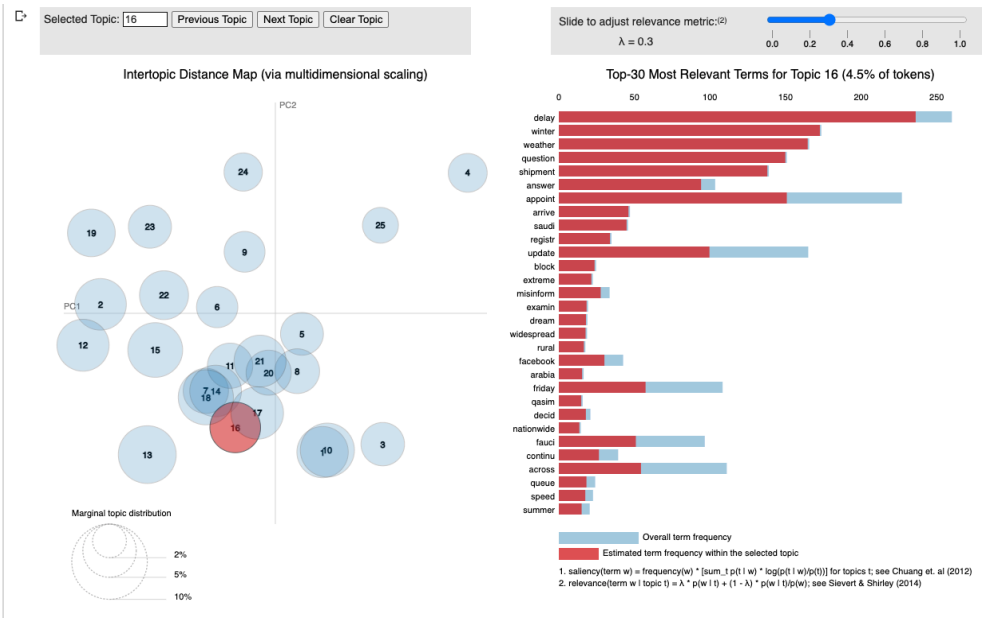

Figure 36: *Topic 16*

As seen in Figure 35, “delay” does have topic-16 as the major contributor. But since topic-8 is quite similar to topic-16, it calls for an inspection.

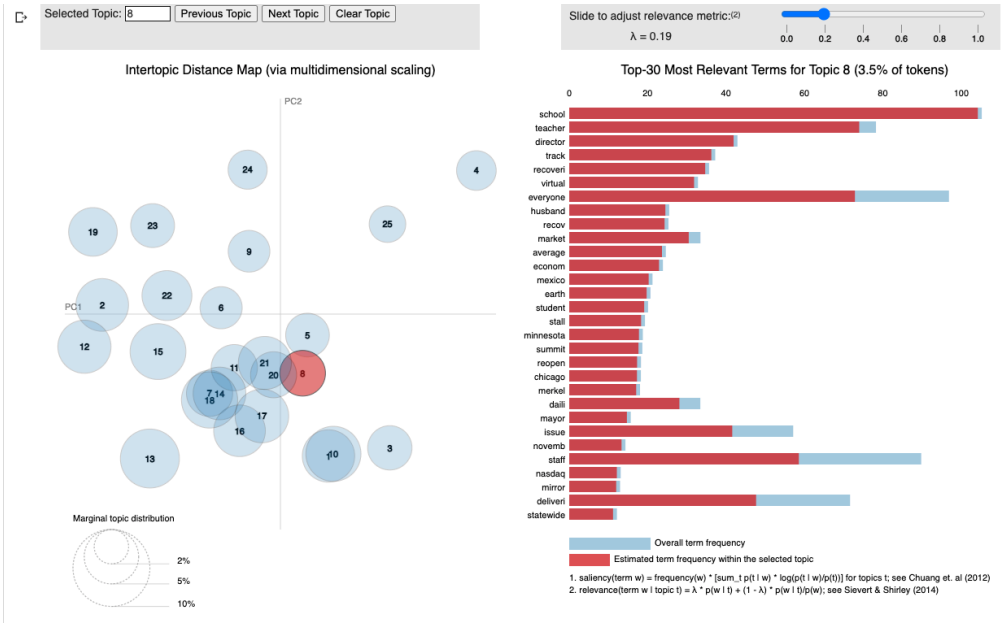

Figure 37: *Topic 8*

High usage of terms like “school”, “staff”, “teacher” and “director” refers to the decision by the Biden administration to vaccinate the schooling staff by the end of March. But we can see the frequent appearance of terms like “everyone” and “recovery” which suggests the public disappointment about “delay” in “reopening” the “market” and cities like “Chicago” from extended lockdowns as the vaccine shipments doesn’t reach the common man.

Geospatial Analysis

Geo-text/ Geotag data offers unique research opportunities through the rich information contained in texts and the special links between texts and geography. As a result, such data facilitates various studies especially those in data-driven geospatial semantics.
Unless and until the origins of such wide variation in willingness to accept a COVID-19 vaccine is better understood and addressed, differences in vaccine coverage between countries could potentially delay global control of the pandemic and the ensuing societal and economic recovery.

For our dataset, we have the location of the tweets. After cleaning up the locations of processed data, we get the coordinates of the tweets with viable locations using OpenStreetMap. These coordinates are then put in the dataframe. For the next step, we use geoPandas to and shapely to convert the dataframe with coordinates to a geo dataframe to form relevant POINTs and POLYGONs to be plotted on the world map on the .shp file.

We get the following plot for our entire dataset; where the green dots represent the location of positive tweets, red dots represent the location of negative tweets and blue dots represent the location of neutral tweets.

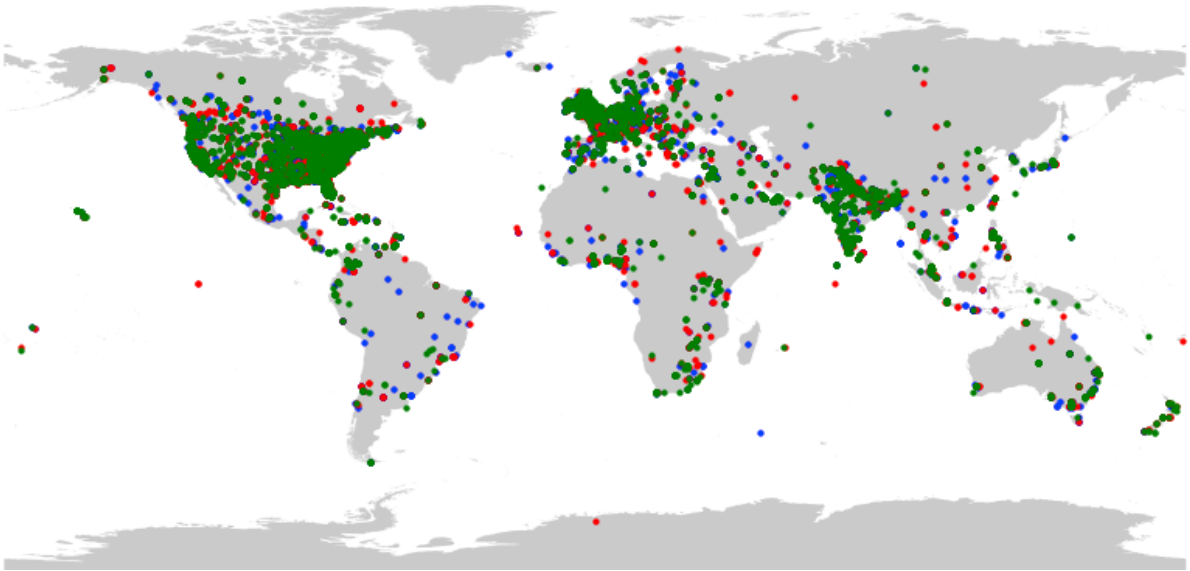

Figure 37: *Geospatial Sentiment Plot*

Insights:

Southern and Eastern Asian countries with a greater number of people being aware about the vaccine and with strong trust in central governments like Japan, South Korea, Singapore and Indonesia had more positive responses. A relatively high tendency toward acceptance in certain parts of middle-income countries, such as Brazil, India and South Africa, was also observed.

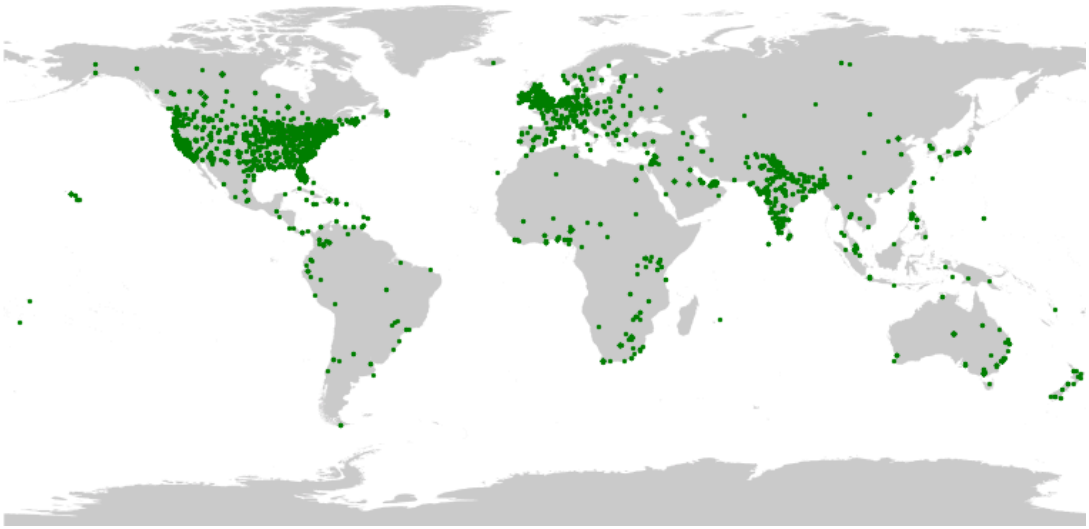

Figure 38: *Positive Geospatial Sentiment Plot*

After giving it a closer look and inspecting the numbers, people of Russia and from the region around Poland and Ukraine had significantly more negative responses. USA had pretty much an evenly divided opinion about vaccination with the Midwest and Texas region being more apprehensive than people living on the either coasts, for example California or New York.

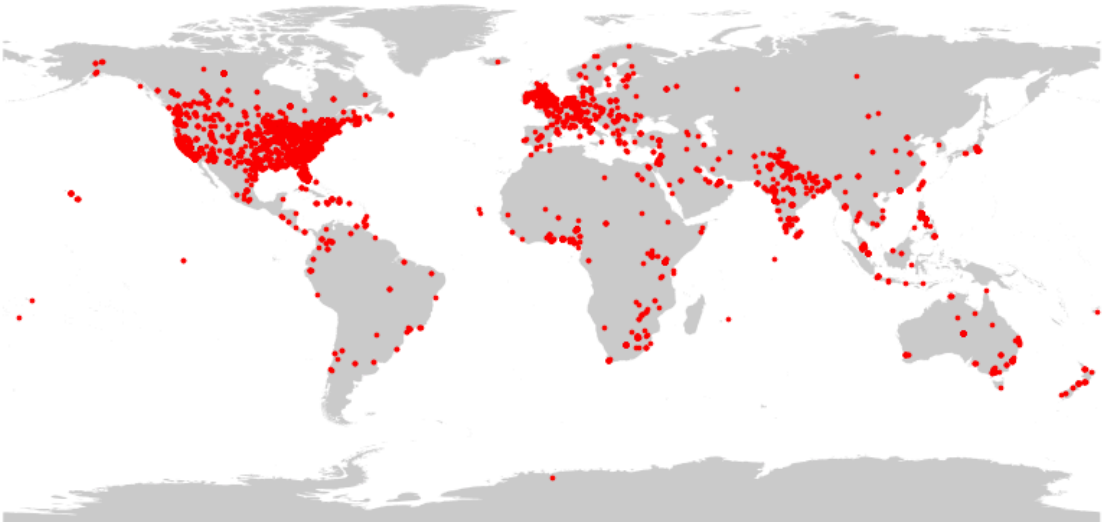

Figure 39: *Negative Geospatial Sentiment Plot*

Innovation and Enhancement: *Classification*

Attention-BiLSTM for aspect-based classification

For enhanced classification, we aimed to perform fine-grained classification using aspect-based sentiment analysis. Aspect-based sentiment analysis (ABSA) is a text analysis method that takes into account categories of data and identifies the sentiment attributed to each category. To incorporate some aspect-based sentiment analysis in our current Bi-LSTM model we added an attention layer to our model.

The basic idea of the modification is to implement an LSTM layer with an attention layer on top to include output of all the LSTM cells instead of basing on just the output of the last cell in the layer. The aim of the attention layer is to merge each word level output to create a sentence level feature to capture global information of the sentence hence allowing more global feature extraction and better aspect classifications. Attention based neural networks implicitly connect aspects with opinions hence improving sentiment analysis results.

Model Overview

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 182, 50)           1228350
_________________________________________________________________
dropout_2 (Dropout)          (None, 182, 50)           0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 182, 128)          58880
_________________________________________________________________
attention (attention)        (None, 128)               310
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 387
=================================================================
Total params: 1,287,927
Trainable params: 59,577
Non-trainable params: 1,228,350
_________________________________________________________________
```

Figure 40: *Model Summary of Attention Bi-LSTM Model*

Model Architecture: Attention Layer

The model architecture of the Attention-BiLSTM model is the same as the Bi-LSTM model described above with an additional attention layer.

- Embedding Layer
- Attention Layer
- Bi-directional LSTM Layer
- Softmax Output Layer

The attention layer calculates a weight vector, and merges word-level features from each time step into a sentence-level feature vector, by multiplying the weight vectors. The attention layer is custom declared with the required operations manually implemented. Tanh is calculated first over the input vector i.e., the output of the bidirectional LSTM layer. Then it is passed to a softmax operation. Finally, the softmax output and the tanh output is multiplied together to get the attention layer output.

The attention layer ensures that not only local information(words) but the global information(sentences) is also considered while training to better capture global information in the tweets. This means that better information and feature extraction is carried out by considering the context of sentences and not only words hence allowing better aspect extraction and categorising. Better aspect categorisation is beneficial when analysing and training on sequential data such as tweets resulting in better performance.

```
class attention(Layer):

    def __init__(self, return_sequences=True):
        self.return_sequences = return_sequences
        super(attention,self).__init__()

    def build(self, input_shape):

        self.W=self.add_weight(name="att_weight", shape=(input_shape[-1],1),
                               initializer="normal")
        self.b=self.add_weight(name="att_bias", shape=(input_shape[1],1),
                               initializer="zeros")

        super(attention,self).build(input_shape)

    def call(self, x):

        e = K.tanh(K.dot(x,self.W)+self.b)
        a = K.softmax(e, axis=1)
        output = x*a

        if self.return_sequences:
            return output

        return K.sum(output, axis=1)
```

Figure 41: *Implementation of Attention Layer in Attention-BiLSTM model*

The above snapshot is the TensorFlow implementation of the attention layer. The attention layer is based on the Bahdanau attention layer. The Attention-BiLSTM uses the same model loss and optimisation as the Bi-LSTM model.

After training the Attention-BiLSTM model for 250 epochs, the following accuracy and loss plots are derived. The validation accuracy value of the model converges to about 0.9031 and the validation loss converges to about 0.3982.

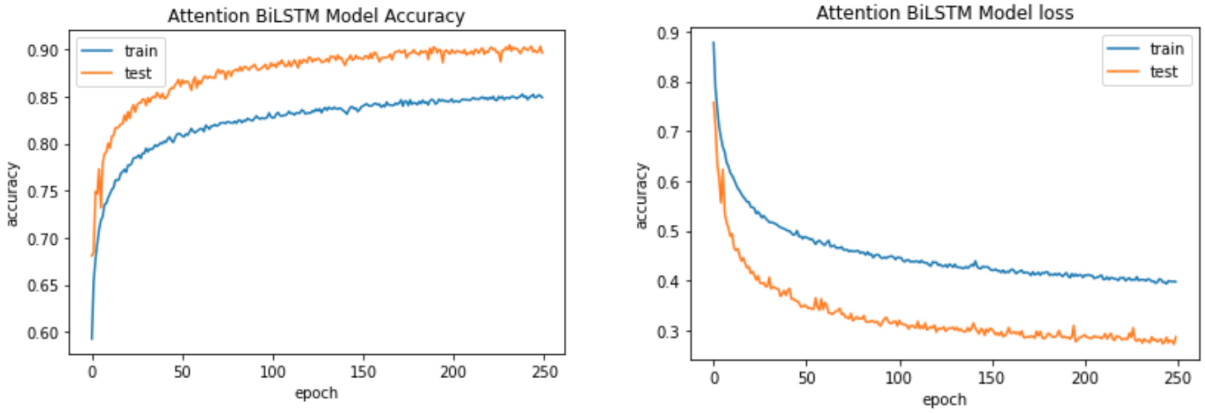

Figure 42: *Model training and validation accuracies and loss plots*

Using the trained model to predict sentiments for the testing data, we achieved a precision score, recall score and F1 score of 0.906. The following confusion matrix and classification report is achieved.

	precision	recall	f1-score	support
positive	0.92	0.85	0.88	1404
neutral	0.89	0.95	0.92	1333
negative	0.89	0.89	0.89	1340
accuracy			0.90	4077
macro avg	0.90	0.90	0.90	4077
weighted avg	0.90	0.90	0.90	4077

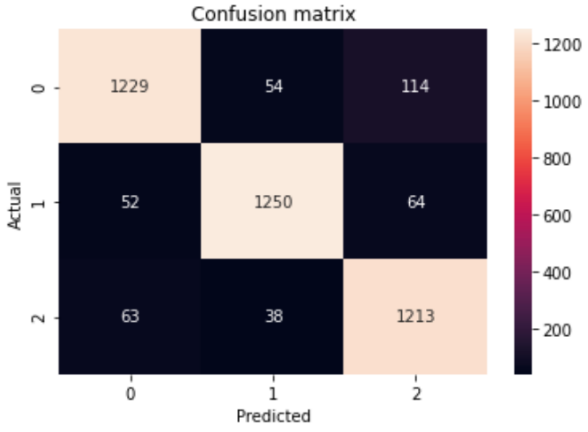

Figure 43: *Classification Report using Attention Bi-LSTM; Confusion Matrix*

Comparison of Results

Label	Bi-LSTM			Attention Bi-LSTM		
	Precision	**Recall**	**F1 score**	**Precision**	**Recall**	**F1 score**
Positive	0.86	0.86	0.86	0.92	0.85	0.88
Neutral	0.88	0.94	0.91	0.89	0.95	0.92
Negative	0.90	0.82	0.86	0.89	0.89	0.89
Weighted average	0.876	0.878	0.877	**0.903**	**0.903**	**0.903**

Table 7: *Result Comparison*

Comparing the results between the base Bi-LSTM model and modified Bi-LSTM model with the attention layer. We can see that the fine-tuning and addition of the attention layer had led to a performance improvement. The weighted average precision, recall and F1 scores have all improved using the attention Bi-LSTM model proving the effectiveness of the aspect-based learning carried out by the attention layer. Labelling of positive and negative sentiments in tweets are also improved using the Attention Bi-LSTM model.

Ensemble Learning: *Random Forest Classifier*

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
In random forests, each tree in the ensemble is built from a sample drawn with replacement from the training set. Furthermore, when splitting each node during the construction of a tree, the best split is found either from all input features or a random subset of size max_features. The purpose of these sources of randomness is to decrease the variance of the forest estimator. Indeed, individual decision trees typically exhibit high variance and tend to overfit. The injected randomness in forests yield decision trees with somewhat decoupled prediction errors. By taking an average of those predictions, some errors can cancel out. Random forests achieve a reduced variance by combining diverse trees, sometimes at the cost of a slight increase in bias. In practice the variance reduction is often significant, hence yielding an overall better model.

Evaluation of Random Forest Classifier

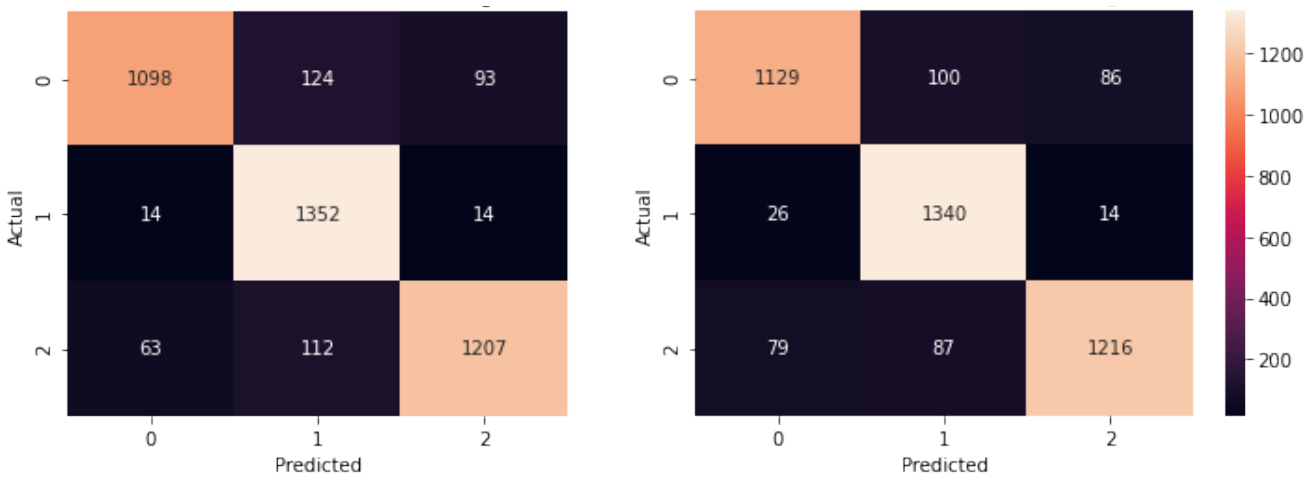

Figure 19: *Confusion Matrix using Countvectorizer (left) and tf-idf vectorizer(right)*

Comparison with a Single Decision Tree:

Model	Count Vectorizer			Tf-idf Vectorizer		
	Precision	**Recall**	**F1 score**	**Precision**	**Recall**	**F1 score**
Decision Tree	0.801	0.798	0.797	0.790	0.790	0.790
Random Forest	0.901	0.897	0.896	**0.905**	**0.904**	**.903**

Table 8: *Result Comparison*

Hyperparameter Tuning: *Grid Search CV*

Hyper-parameters are parameters that are not directly learnt within classifiers. They are passed as arguments to the constructor of the classifier classes. It is possible and recommended to search the hyper-parameter space for the best cross validation score. Any parameter provided when constructing a classifier may be optimized Grid Search.

Grid Search implements an exhaustive search over specified parameter values for a classifier. The parameters of the classifier used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

We implemented grid search to tune hyperparameters of various classifiers like SVM, KNN, Decision Trees and Random Forest. Detailed implementation of this search for hyperparameter tuning can be found in the attached code file.

Evaluation Metrics: *K-Fold Cross Validation*

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model. It generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

Model	Count Vectorizer			Tf-idf Vectorizer		
	Precision	Recall	F1 score	Precision	Recall	F1 score
Naive Bayes	0.726	0.693	0.691	0.736	0.720	0.720
KNN	0.642	0.558	0.545	0.687	0.654	0.652
SVM	0.800	0.794	0.792	0.800	0.799	0.799
Decision Tree	0.647	0.626	0.622	0.660	0.646	0.643
Random Forest	0.791	0.751	0.746	0.795	0.783	0.783

Table 9: *Summary of Evaluations using 5-fold Cross Validation*

Conclusion

For our project, we have crawled the tweets for the response to the COVID-19 vaccine. To scrape query-specific Twitter data, snscrape was used. We then created a web interface, consisting of HTML and PHP files, and integrated them with the functionalities of the indexing software, Solr, where the required data manipulation, indexing and selection are carried out. After which, we proceed with data pre-processing using techniques such as merging and formatting the multiple CSV files of crawled data, removal of duplicated data, stemming, lemmatization, stopwords removal and count vectorization.
Following that, we performed our sentiment prediction using various classification models such as Naïve Bayes classifier, K-nearest Neighbour and Support Vector Machine. The models are then evaluated using evaluation metrics such as F-measure, precision and recall values and the results are compiled and analysed. The powerful LDAvis library along with geoPandas is used to visualize text data even with geotags without much difficulty. Lastly, we explored some innovations to enhance our classification such the use of GridSearch and k-fold Cross Validation and the use of ensemble classification model like Random Forest to compare with the models' results.

In a situation like this where the world is trying to stand back up after this pandemic took the world by storm, analysing the global response towards vaccine becomes critically important for governments and health organisations. We see our project's relevance and usage making it a vital cog in the wheel for today's world.

Links: *Video, Data and Source Code*

Resource	URL
Video (YouTube)	*https://youtu.be/qfNdTsCS6kU*
Data and Source Code (Google Drive)	*https://drive.google.com/drive/folders/1lHAQDIAwguGcYqqSC7ByPHrydcqLKRTn?usp=sharing*

Table 10: *Guide for links to Video, Data and Source Code*