Minerva: Opinion Mining of Tweets about E - Learning using Self - Organizing Maps

A Thesis Project Presented to the Faculty of

School of Computer Studies

University of San Jose-Recoletos

Cebu City, Philippines

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Computer Science

By

La Rosa, Rhyan Augustine D.

Legaspi III, Redempto D.

Lopez, John Paul B.

Mrs. Marisa M. Buctuanon

Adviser

June 2022

TABLE OF CONTENTS

ACKNOWLEDGEMENT

To our family and friends who have wholeheartedly supported us throughout our academic journey. They gave us the courage and motivation to persevere through hardships.

To the School of Computer Studies who have continuously held our hand during our endeavors. They are the ones who provided us with the necessary knowledge and skills to finish this research paper and become the skilled professionals that we are today.

Special Mentions:

Mrs. Marisa M. Buctuanon, who guided us throughout the project. Her insights and expertise have given us the direction we needed to finish the project.

Dr. Gregg Victor Gabison, who showed full support for our project. His enthusiasm and optimism for our project gave us the confidence to continue with the project.

Dr. Jovelyn Cuizon, who gave her critical and valuable advice. She provided amazing insight on how we could improve the project.

Mr. Eric Magto, who has been there since the beginning. He has given us help and guidance even during his free time.

Mrs. Josephine E. Petralba who gave us her insights on how to properly write the manuscript and motivate the team to continue their goal and go after the defense.

ABSTRACT

This study aims to provide a new tool for schools, universities, government bodies, and other similar organizations to gather the general opinion of students and teachers on online classes, a new form of teaching popularized by the COVID-19 pandemic. The researchers achieve this by gathering their data through Twitter and utilizing Self-Organizing Maps to categorize the tweets into different topics. These topics would then be the different aspects of an online class that have a significant impact on the students. Afterward, sentiment analysis is performed to interpret whether the opinion was terrible or good regarding the topic it belongs.

CHAPTER 1

INTRODUCTION

Rationale of the Study

Online classes are an innovation in the field of education that was necessitated by the COVID-19 pandemic. Although it has been around before the virus outbreak, it has only become widespread in the year 2020. Students of all ages and nationalities are now partaking in a new form of teaching that not even the teachers are prepared for. It is still very new to both students and teachers alike. This means that online classes will also come with a different set of problems previously not experienced before in an educational environment, such as getting the opinion of both students and teachers alike on online learning. The study aims to extract the opinion of both these actors and identify their grievances.

Twitter is a medium where students can express their opinions honestly and conveniently. Most articles that cover Twitter as a way for opinion mining use Random Forest or Logistic Regression, such as in the work of Alcober and Reveno[1]. K - Nearest Neighbors and Naive Bayes, which used in the work of Mujahid et al.[2] There has been no current use of the Self-Organizing Map for topic modeling on Twitter. This research offers a novel way to classify tweets and get alternative results with a different algorithm.

This study aims to use Self - Organizing Map as a way to do topic modeling on the social media application Twitter. The user can get topics based on the scraped data, then apply to chunk to separate the sentences or phrases on the tweet with different meanings or topics. Finally, the users would get a graphical representation of the results of the sentiment analysis model and their related topics.

Theoretical Background

Analyzing high-dimensional data can be very difficult because the researchers can only visualize data in 2 or 3 dimensions. Before and after training machine learning models on large high-dimensional datasets, it is desirable to have some intuitive understanding of the relationships and patterns in a dataset to guide the learning process and help interpret results.[3] The Self-Organizing Map was a viable alternative to more traditional neural network architectures.[4] This study focuses on opinion mining of tweets of a SOM model as a way to acquire topic models on a Twitter corpus. In our procedure, Self - Organizing Map uses large amounts of data such as web scraped tweets from Twitter to provide better visualization and classification. Data classification is about categorizing and organizing data for better analysis and decision-making [5]. A topic modeling system based around SOM firstly relies on a vast data set that is cleaned and tokenized to effectively learn the ideas used in the data, which is helpful on our end since the researchers will use scraped data from Twitter.

Web scraping is using bots to extract content and data from a website. Unlike screen scraping, which only copies pixels displayed on the screen, web scraping extracts underlying HTML code.[6] The scraped data will then be stored in the database for preprocessing or data cleaning.

Data cleaning is fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct[7]. The resulting scraped data may contain filler words, tweets with non-English words, links, and audio or video tags that contain little to no meaning to our model. The resulting pre-processed data will then be given a numerical representation called vectorization.

Vectorization is a step in feature extraction. The idea is to get some distinct features out of the text for the model to train on by converting text to numerical vectors[8]. It is imperative for processing textual data in natural language processing applications. It enables the machines to understand the textual contents by converting them into meaningful numerical representations[9]. TF-IDF or Term Frequency - Inverse Document Frequency is one of the most used vectorization techniques in natural language processing. It can be separated into two parts. Term frequency is where the

total times a term appears in a document is divided by the total count of words in a document. Inverse document frequency looks at how common (or uncommon) a term is amongst the corpus[10]. The values of TF and IDF are multiplied to create the TF - IDF vectors, where TF gives us information on how often a term appears in a document, and IDF gives us information about the relative rarity of a term in the collection of documents[11].

The vectorized matrix will be used to train the SOM model. The SOM model is a matrix of N rows and M columns containing an array of weights to represent the features of the data. Its process includes taking a random row from the TF-IDF matrix and using it to update the weights of every cell in the SOM matrix. In this case, every column in the TF-IDF is a feature that describes the document.

To optimize the topic model, the researchers will use Topic Coherence, where each coherence of the topic is assessed. Coherence measures have been proposed in the NLP community to evaluate topics constructed by some topic models. In a more general setting, coherence measures have been discussed in scientific philosophy as a formalism to quantify the hanging and fitting together of information pieces[12]. The topics generated by the SOM will be used to get the topic coherence of the model. Umass coherence scores are based on the document co-occurrence counts with logarithmic conditional probability as a confirmation measure[13]. Since Umass uses the co-occurrence probability from the existing corpus and does not need to depend on

further gathering of an external corpus[13], it is the best fitting topic coherence model for our algorithm. Chunking will then be applied to the Tweet to separate the sentences with different topics and sentiments. Finally, use Valence Aware Dictionary and Sentiment Reasoner (VADER), which is a rule-based and lexicon tool to get the sentiment scores. VADER is used because it has been attuned explicitly to analyzing sentiments expressed in social media[14]. The placements of documents in the clusters will be measured by the silhouette score, where the best possible value relating to the document distance is one, and the worst possible value is -1. Values near 0 indicate documents with overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar[15].

Review of Related Literature

Using Self - Organizing Maps to get topic models is uncommon mainly, especially in the Philippines. The following studies use Self - Organizing Map to get a conclusion based on a vast array of inputs and also other data mining algorithms used for Twitter.

In Akshay Kapoor and Veni Jindal's research work[16]. Self - Organizing Map was used to reduce the dimensionality and visualize the polarity of sentiment in tweets. Their approach was using SOM as a way to display large amounts of data from Twitter. They

also used Sentence BERT(SBERT), a sentence embedding process using a Siamese and triplet network. This helps them find the most similar documents.

In Relia et al.'s work[17]. Self - Organizing Map was used to cluster and map different tweets from all over New York City and identify what regions were susceptible to racist and homophobic slurs. Their embedding algorithm was similar to Word2Vec but trained on actively learned racism and homophobia data sets. The embedding model is an external, two-layer neural network that is trained to reconstruct the linguistic context of words from its training text corpus input. The model produces a high-dimensional vector space, with each unique word in the corpus being represented by a vector in the space. The model is then validated by comparing the performance to a baseline classifier, mainly Support Vector Machines (SVM)[18].

In research by Nimasha Arambepola[19], tokenization, removal of stop words, and lemmatization were used to clean the dataset and remove noise. TF - IDF with TextBlob was used to extract the most relevant topics on Twitter. After assigning the scores for all the words in the dataset using TF - IDF, TextBlob was used to get the polarity and subjectivity of the sentiments.

Significance of the Study

This study provides the students, administrators, and researchers with an in-depth look at the impact of online classes on teachers, parents, and students. Providing an opinion on online classes without the need to conduct a survey and people tend to express their opinions more on social media since it can be somewhat anonymous, convenient, and quick responses.

As of the moment, there are no articles that use Self Organizing Map as a way to get models in the Philippines. This study can act as a medium for both students and teachers alike. It can provide a form of constructive criticism of the school based on the students' opinions without the need to directly ask them while also providing a different perspective than the conventional algorithms.

Project Objectives

This research aims to create a topic modeling system that can help educators, students, and school administrators are informed about the present sentiments of people engaged with online learning on Twitter. The objectives of this project are:

1. Scrape Tweets from Twitter
2. Apply data processing techniques to the scraped Tweets
3. Apply vectorization with Term Frequency - Inverse Document Frequency on the processed Tweets

4. Apply topic modeling using the Self-Organizing Map algorithm
5. Perform chunking, then apply sentiment analysis based on the resulting topics
6. Hyperparameter tuning of the Self-Organizing Map algorithm using Topic Coherence
7. Get the silhouette score of the Self-Organizing Map model for testing
8. Develop a mechanism to associate new tweets to an appropriate cluster and get its sentiment

Project Scope and Limitations

The study is mainly focused on English tweets for the entire country of the Philippines, the pandemic and how they adapted. The main limitation of this study is that the researchers cannot cater to other languages, such as tweets in Tagalog, Bisaya, Waray, and other languages from the Philippines, due to them lacking a significant amount of corpus supported by the community. The researchers also cannot verify if the tweet was sarcastic or if the user is an active student or a teacher. This wrong data can be prone to fake news as everyone can express their opinion on a specific topic without validation leading to the tarnishing of the name of a particular school or institution.

The tweet scraper also has limitations in scraping data online. The researchers are limited to using 20 keywords only. Another limitation is that the researchers can only scrape tweets posted seven days before the set scrape date since Twitter requires a paid service for historical tweets. Specific tweets involve wrong grammar, wrong spelling, or tweets having mixed languages which leads to unavoidable noise within our scraped data.

In the pre-processing part, the noise still passes through, such as non-English tokens. Although the researchers filtered English tweets, some tweets contain non-English tokens, and these are hard to pluck individually. A possible method would be to either translate the non-English tokens or remove them altogether. None of these worked in our study because of the lack of accessible and accurate language packages. The SpaCy package cannot correctly detect the language of a single word and can only detect a complete sentence. Meanwhile, google trans, text blob, and even the official Google Cloud translation API limit how many free requests are allowed. Hence, the researchers decided to let non-English tokens pass through as it is unavoidable.

Research Methodology

This research utilizes the ability of a Self - Organizing Map to perform Topic Modeling based on the scraped Tweets from Twitter. The most identical tweets will then be grouped within the SOM.

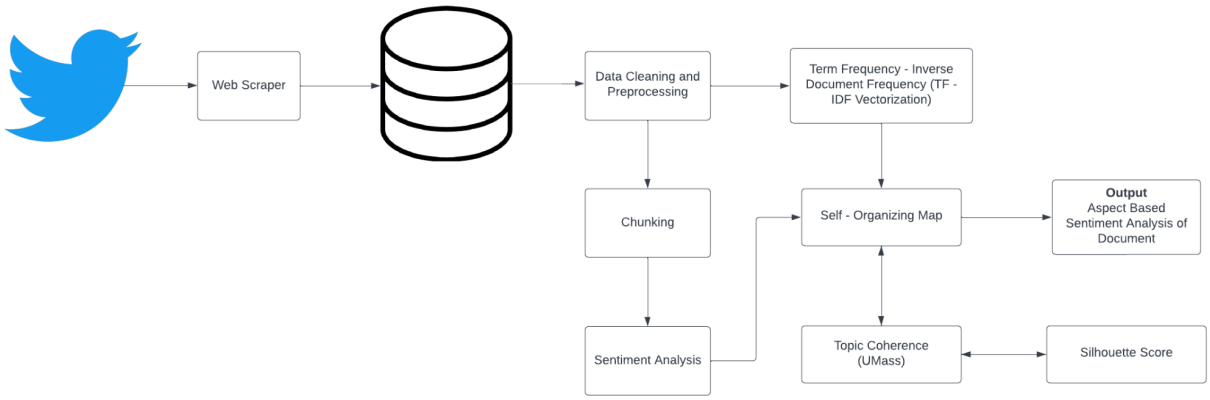


Figure 1: Use Case Diagram

Figure 1 shows the whole process of the system from the data being scraped from Twitter. Then it is stored in the database to be cleaned and pre-processed. Term Frequency - Inverse Document Frequency(TF - IDF) vectorization will then be applied to the pre-processed data. The resulting vectorization matrix will be used by the Self-Organizing Map(SOM) to produce an (n x m) matrix where each node is equal to the num_features sent from the TF – IDF; the matrix is then normalized from an n by m times the num_features to a ((n x m) x num_features). The resulting topics will be used by the sentiment analysis and the chunker to separate the sentences or phrases with different meanings.

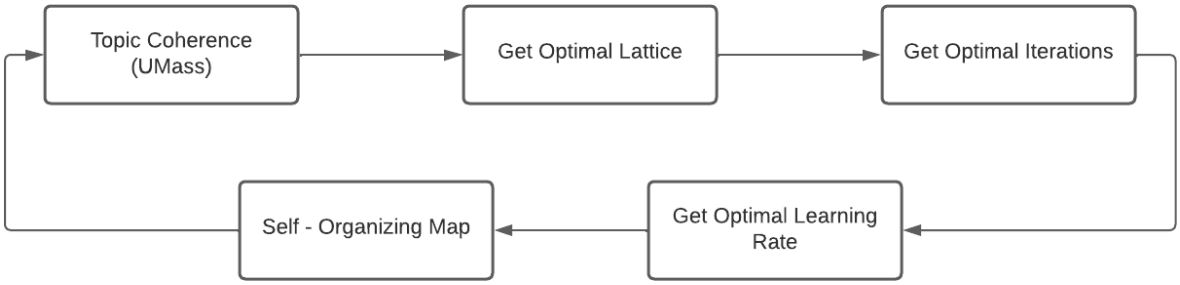


Figure 2: Topic Coherence Use Case Diagram

Figure 2 above shows the detailed diagram of the topic coherence for the SOM model. The model is fed to the topic coherence with a UMass calculation. The hyperparameter tuning for the SOM model will be used, such as the lattice size, optimal iterations, and the learning rate.

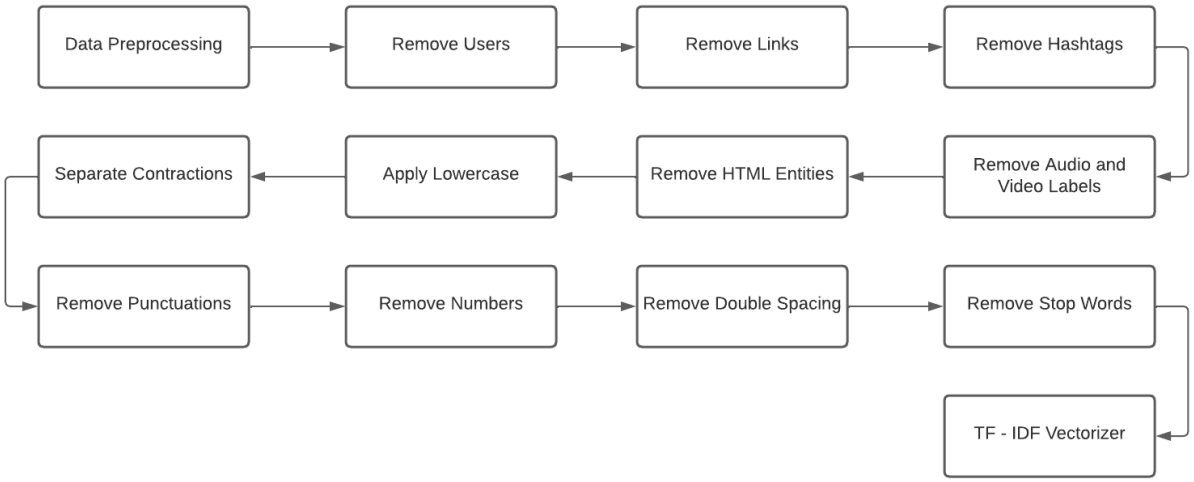


Figure 3: Data Preprocessing Use Case Diagram

Figure 3 shows us the flow of the data pre-processing from the tweets scraped from Twitter leading to the TF - IDF Vectorizer. The pre-processing will be used to prevent noise in the corpus.

CHAPTER 2

SOFTWARE REQUIREMENTS AND DESIGN SPECIFICATION

Use Case Diagram

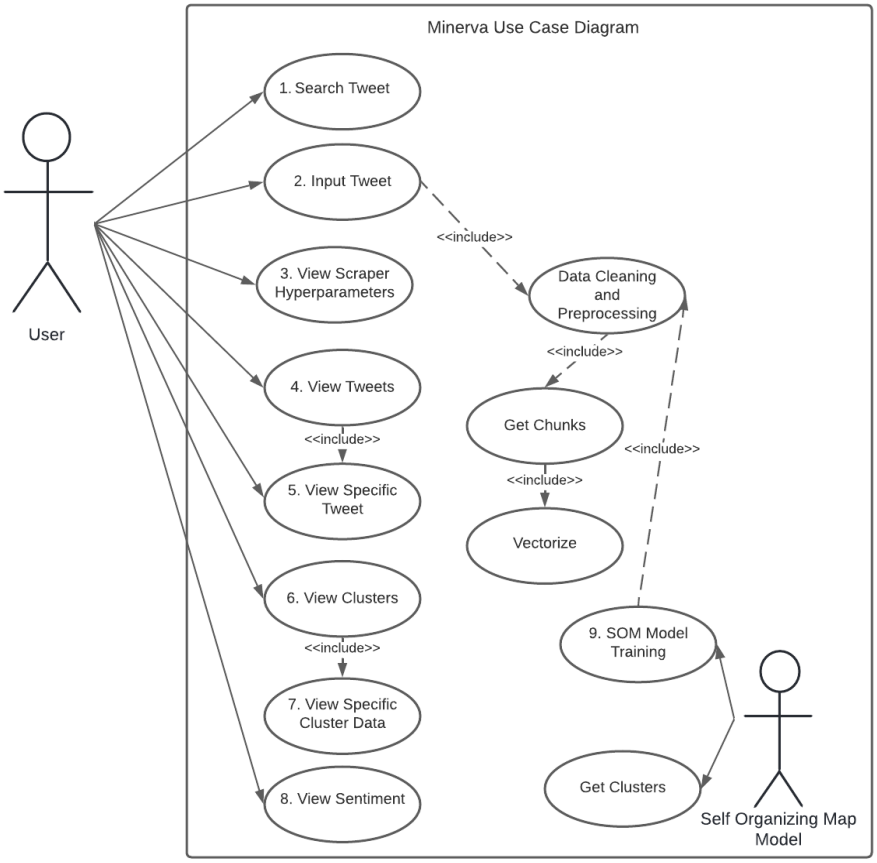


Figure 4: Use Case Diagram

Figure 4 shows the use case diagram describing the interaction between the user and the system. The uncleaned tweets will then undergo pre-processing, which results in removing stop words, pruning of tweets, and applying n-grams. The resulting cleaned tweets will then be stored back in the database. The cleaned tweets will undergo Term Frequency - Inverse Document Frequency for designation of weights generating a document to word matrix, which will be used to generate a Self Organizing Model. The user can adjust the parameters of the SOM. The user interface will display the resulting clusters and the sentiments based on the tweets. The user interface will also display the uncleaned and cleaned tweets from the database.

Use Case Narratives

The functional requirements show the use case narrative of Minerva; it highlights the interactions between the user and the system in a more descriptive manner. The tables below show the use case functions of the system.

1. **Search Tweet**

The table below shows how the user can search for a keyword, hashtag, or date range in the raw tweets list. The search will be handled by the web application to get the tweets under the search input.

Use Case	Search Tweet

Actors	User
Type	Non - Essential
Precondition	The user has pressed the display tweets tab
Postcondition	Displays the tweets which satisfies the search inputs of the user
FLOW OF EVENTS	
Actor Action	System Response
1. The user opens the display tweets tab	3. The frontend will sort all the raw tweets list based on the search queries
2. The user enters desired search query	4. The system will display all the results of the search in a list

Figure 5: Search Tweet

2. Input Tweet

The table below shows how the user can manually input a tweet and then will be analyzed to the SOM model. After the analysis, the cluster and sentiment will be returned for viewing.

Use Case	Input Tweet
Actors	User
Type	Non - Essential
Precondition	The user has launched the web application
Postcondition	The user gets a feedback notification that the tweet has been stored
FLOW OF EVENTS	
Actor Action	System Response
1. The user inputs a tweet	2. API controller handles the request and gets the SOM model
	3. Tweet undergoes chunking, pre-processing, and vectorization.
	4. SOM model determines the cluster of the new tweet.
	5. Sentiment analysis is applied to the new tweet.

	6. Clusters and sentiments belonging to the new tweet are displayed

Figure 6: Input Tweet

3. View Scraper Hyperparameters

The table below is optional to the user; they can see the details of the hyperparameter for the scraper. The hyperparameters of the scraper come from the raw tweets table of the database.

Use Case	View Scraper Hyperparameters
Actors	User
Type	Non - Essential
Precondition	The user has launched the web application
Postcondition	Scraper hyperparameter is displayed
FLOW OF EVENTS	
Actor Action	System Response

1. The user opens the dashboard tab	2. API controller will handle the request and query the tweets table
	3. The system will display the parameters of the scraper

Figure 7: View Scraper Hyperparameters

4. View Tweets

The table below shows that the user can view a list of scraped tweets once the user clicks on the display tweets tab. The web application will display all the raw tweets stored in the database for viewing.

Use Case	View Tweets
Actors	User
Type	Non - Essential
Precondition	The user has pressed the display tweets tab
Postcondition	Displays the uncleaned tweets
FLOW OF EVENTS	

Actor Action	System Response
1. The user opens the display tweets tab	2. API controller will handle the request and open the tweets table in the database.
	3. The system will display all the tweets in the raw tweets table of the database.

Figure 8: View Tweets

5. View Specific Tweet

Once the user clicks on a list of tweets, they can view a specific Tweet. Complete detail of the Tweet will be displayed, with the raw tweet, cleaned tweet, date of Tweet, the list of TF - IDF values, and list of related clusters.

Use Case	View Specific Tweet
Actors	User
Type	Non - Essential
Precondition	The user has pressed a particular tweet
Postcondition	Displays the specific tweet
FLOW OF EVENTS	

Actor Action	System Response
1. The user opens the display tweets tab	2. All uncleaned tweets will be posted on the user interface
3. The user has pressed a particular tweet	4. The user interface will display the specific tweet

Figure 9: View Specific Tweet

6. View Clusters

Once the SOM model is finished building, the user can view the clusters generated by the model. All tweets inside the clusters will be separated according to their sentiments, and the results will be displayed.

Use Case	View Clusters
Actors	User
Type	Non - Essential
Precondition	The user has pressed the display clusters tab
Postcondition	Displays all the clusters generated by the SOM model

FLOW OF EVENTS	
Actor Action	System Response
1. The user opens the display clusters tab	2. All clusters will be displayed in a list, and their respective documents

Figure 10: View Clusters

7. View Specific Cluster Data

The table below shows how the user can get a list to view the specific clusters the Tweet belongs to. This can be used if there is a trained SOM model in the database.

Use Case	View Specific Cluster Data
Actors	User
Type	Non - Essential
Precondition	The user has chosen a specific cluster from the list
Postcondition	Displays all the documents belonging to the specific cluster
FLOW OF EVENTS	

Actor Action	System Response
1. The user opens the display clusters tab.	2. All clusters will be displayed in a list, and their respective documents.
3. The user chooses a specific cluster from the list.	4. The user interface will display all the documents belonging to the specific cluster.

Figure 11: View Specific Cluster Data

8. View Sentiment

The table below shows how the user can get a detailed view of the entire sentiment of the Tweet. The system will prompt the API to request details on the sentiment table. The bar graph will display different sentiments of the month and chosen year.

Use Case	View Sentiment
Actors	User
Type	Non - Essential
Precondition	The user has pressed the dashboard
Postcondition	Displays a bar graph of all documents

	and their sentiments
FLOW OF EVENTS	
Actor Action	System Response
1. The user opens the display sentiment tab.	2. Controller prompts the API for stored sentiments in the database.
	3. A bar graph is shown based on the different sentiments of the tweets, and it is sorted by year.

Figure 12: View Sentiment

9. SOM Model Training

The table below shows a detailed flow of events for the SOM model training. The use case is triggered by the administrator, where the system will get all the training data and then create a TF - IDF matrix for the SOM model to train.

Use Case	SOM Model Training
Actors	SOM Model
Type	Essential
Precondition	There must be an existing training data in

	the system
Postcondition	Topics are generated with their clusters
FLOW OF EVENTS	
Actor Action	System Response
1. The model starts SOM training.	2. Training data is loaded into the TF - IDF for matrix generation.
	3. Hyperparameters such as lattice size, iterations, and learning rates are set.
	4. The trained SOM model is stored in the database for accessing and later use.

Figure 13: SOM Model Training

Activity Diagram

Search Tweet

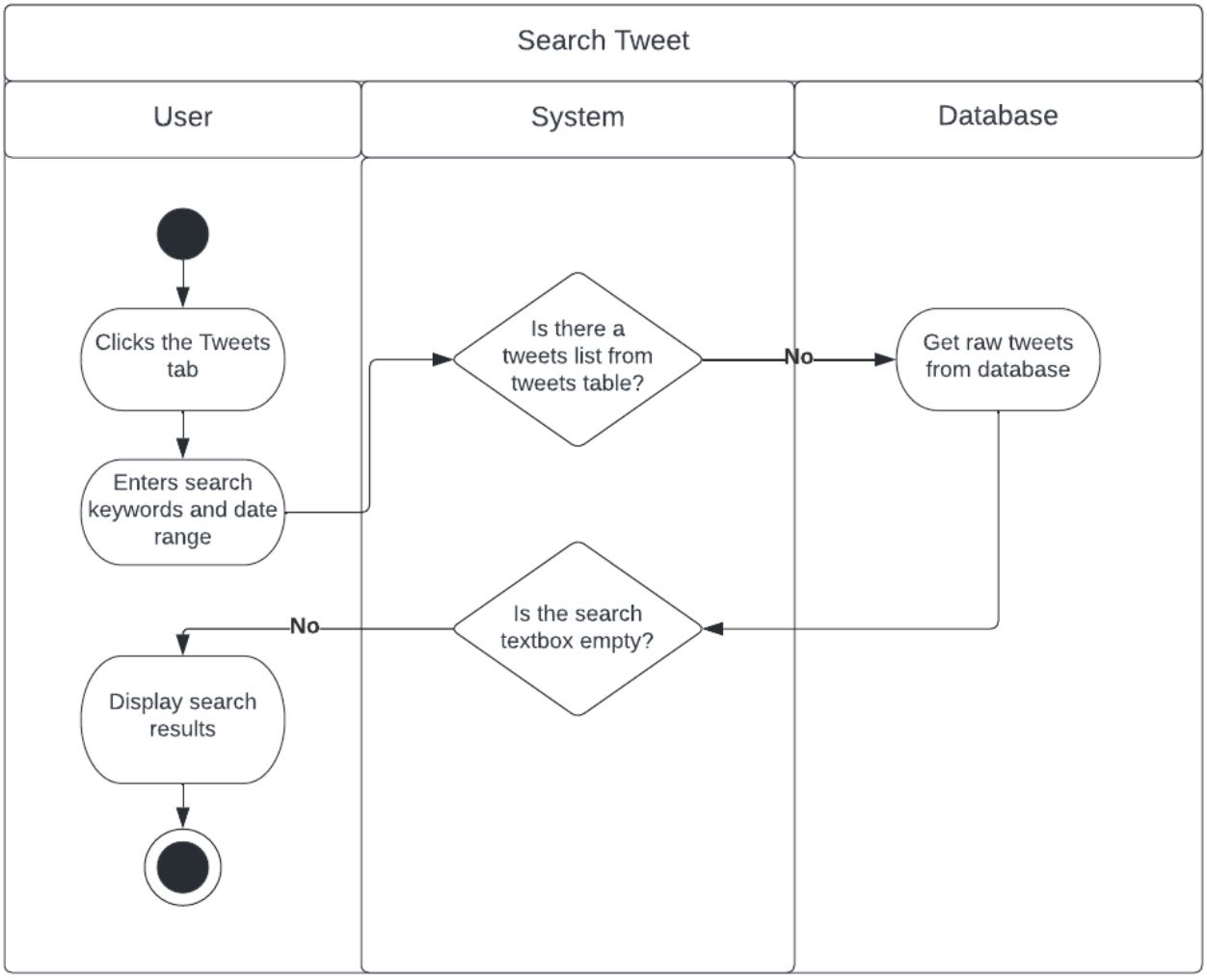


Figure 14: Search Tweet Activity Diagram

The user enters a set of search queries, whether in the search box for keywords and hashtags or the date range to display the raw tweets. The raw tweets list will first be fetched from the database and sorted within the frontend web application.

Input Tweet

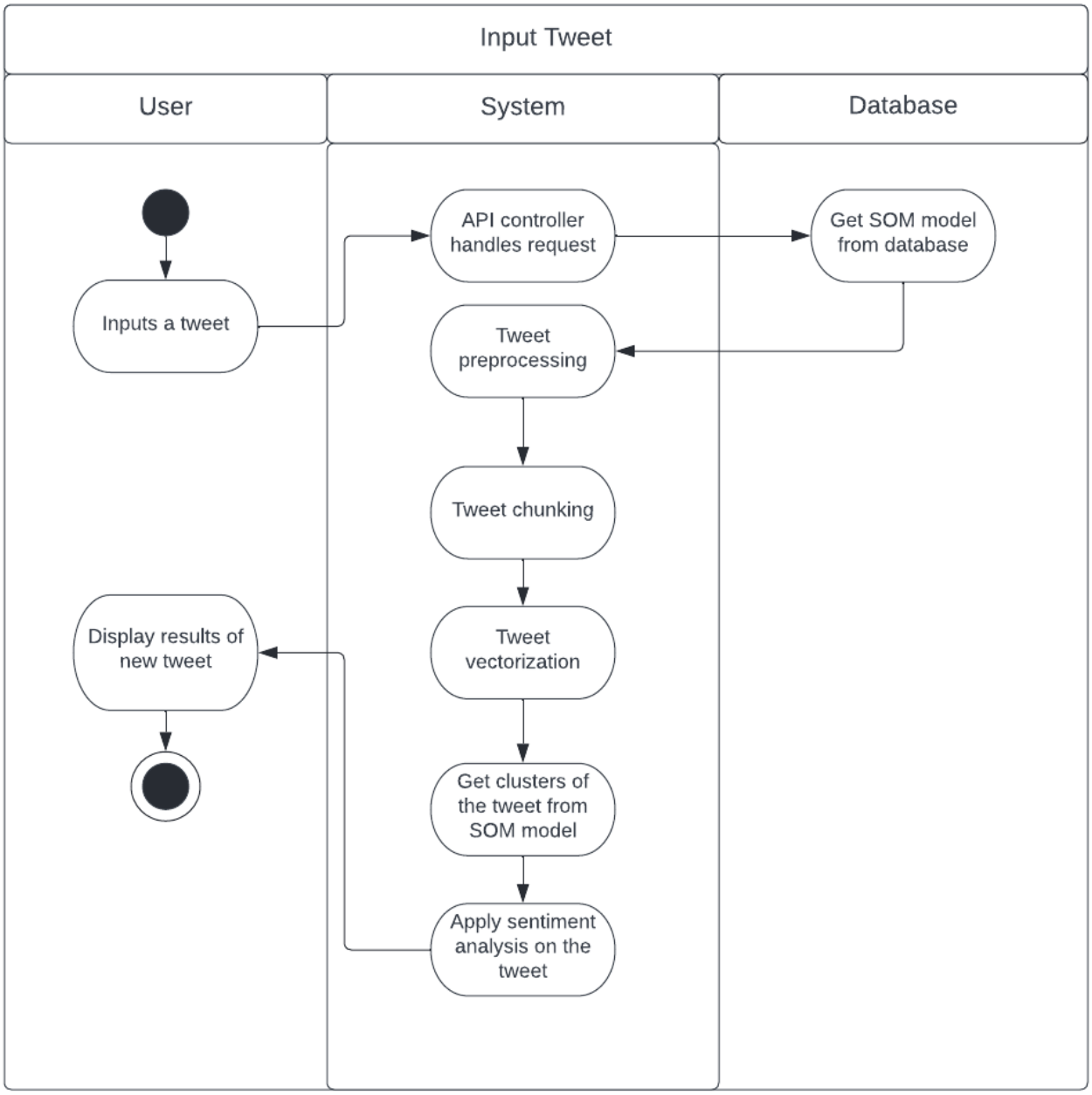


Figure 15: Input Tweet Activity Diagram

Once the user enters a tweet, the API controller will handle the request and get the existing SOM model from the database. The new tweet will undergo the conventional process of a scraped tweet, which is to apply pre-processing, chunking, TF

- IDF vectorization, get the clusters of the tweet on the SOM model and apply sentiment analysis. The results will be displayed on the web application for viewing.

View Scraper Hyperparameters

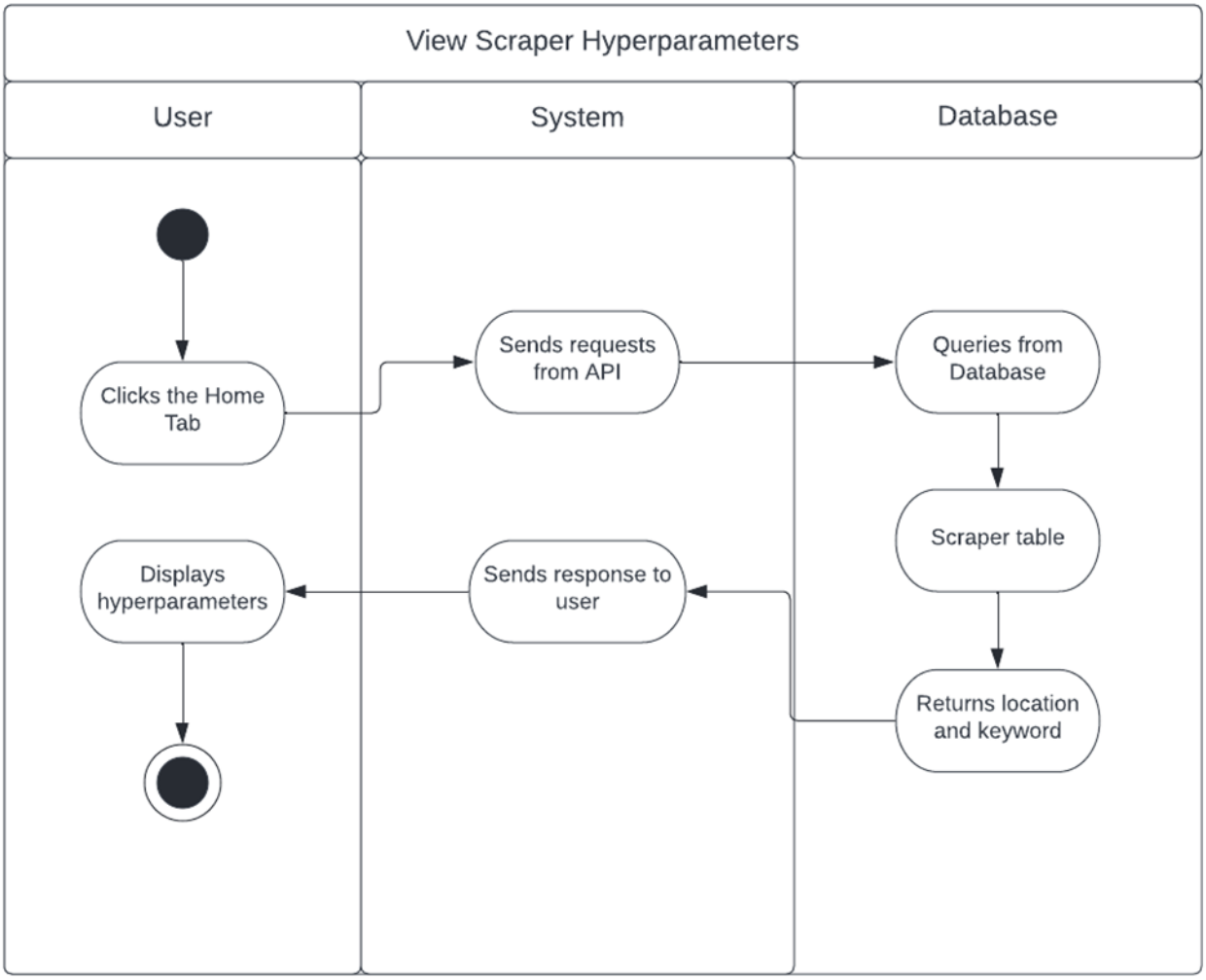


Figure 16: View Scraper Hyper Parameters Activity Diagram

Figure 16 shows the hyperparameters of the scraper in detail. Once the user clicks the home tab, the system will send a request to the API to find queries from the database if there are any saved parameters. Then the database will return the scraper details, and then the system will display it to the user.

View Tweets

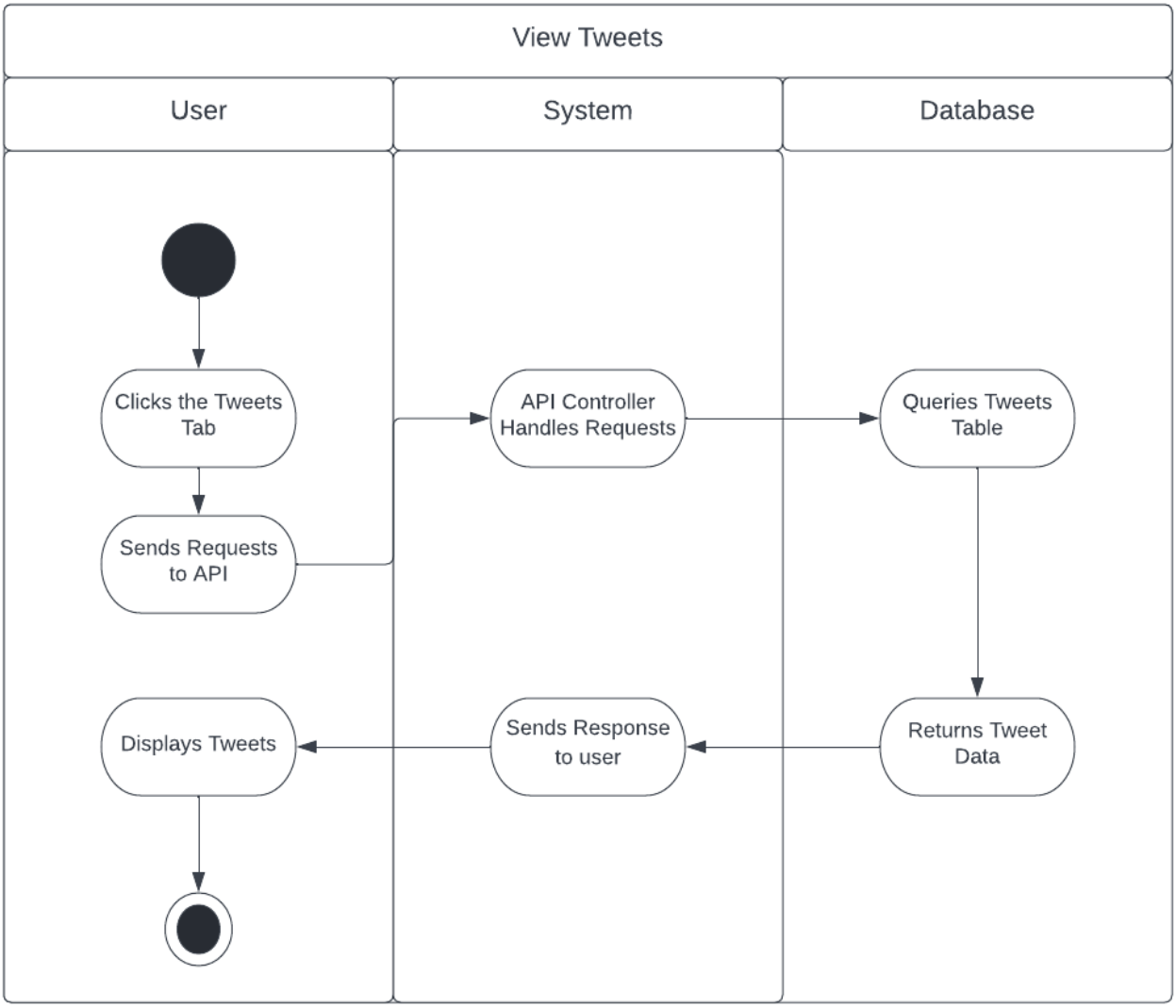


Figure 17: View Tweets Activity Diagram

Figure 17 shows that once the user clicks the Tweets Tab, the system will send requests to the API, then the API will handle the request. The database will then query the Tweets table to get the complete list of Tweets, then return the list of Tweets data, and then the system will display the Tweets.

View Specific Tweet

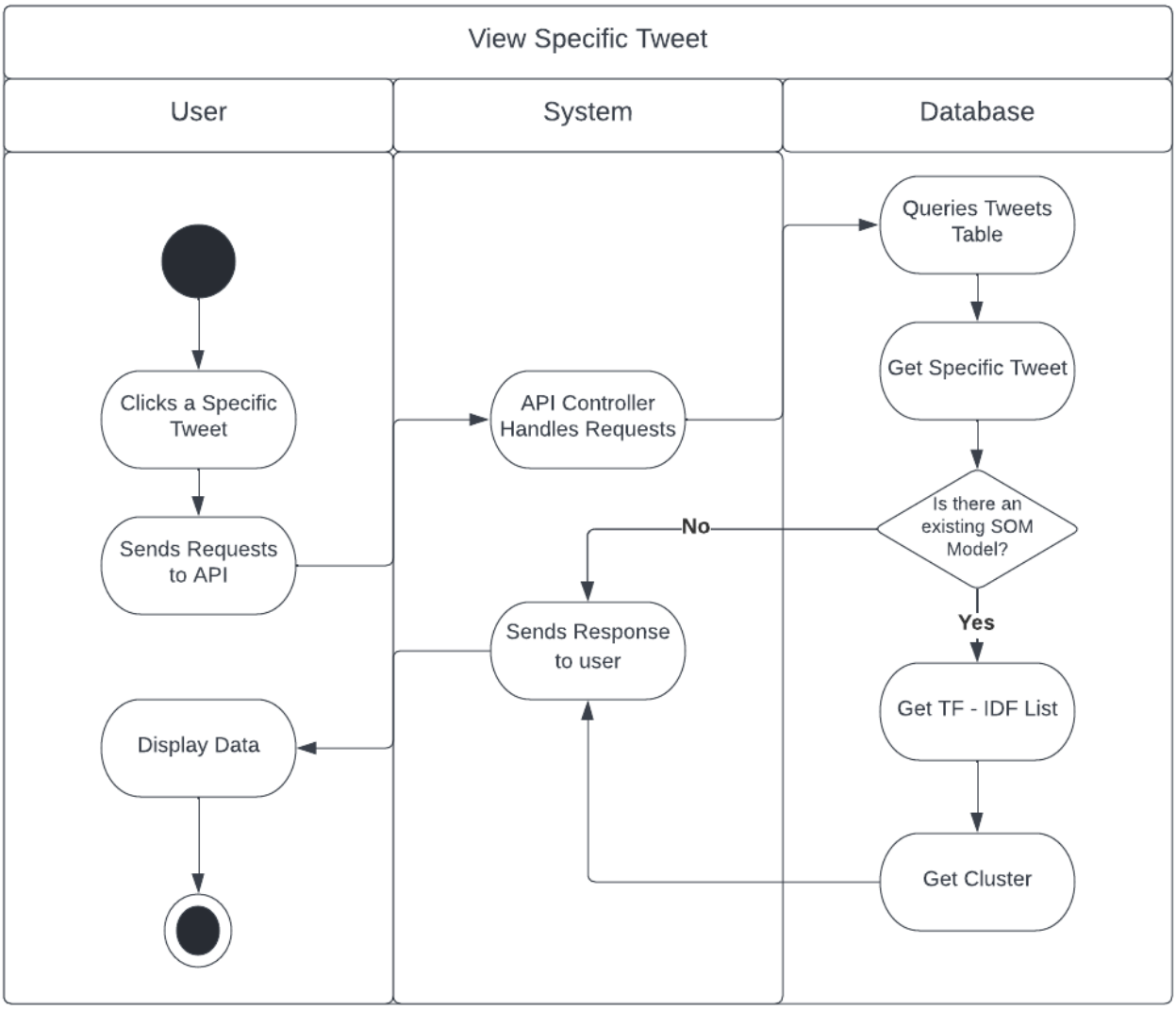


Figure 18: View Specific Tweet Activity Diagram

Figure 18 shows the flow of a specific Tweet in detail. When the user clicks a specific Tweet, the system will send a request to the API. The API will then handle the request and queries the Tweets table based on the tweet_id, which is a unique key to the Tweet. It will then check if there is an already built SOM model, if there is no model, it will just directly display the Tweet without the TF - IDF list and cluster; if there is an

existing model, it will get the TF - IDF list of the specific Tweet and then get its relating cluster.

View Clusters

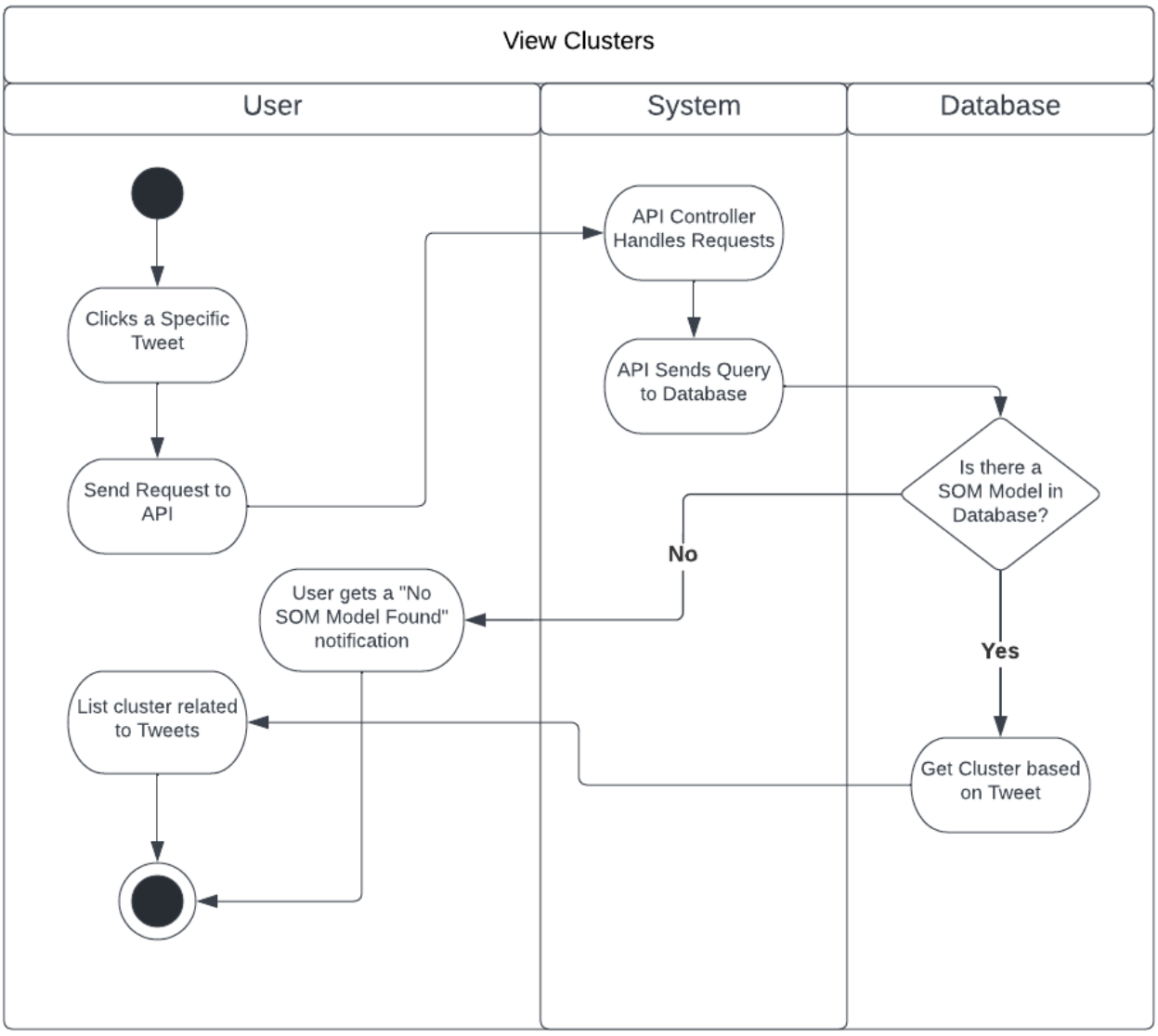


Figure 19: View Clusters Activity Diagram

Figure 19 shows the continuation of Figure 18, which is View Specific Tweet. Once the user clicks a specific Tweet, it will send a request to the API, then it will handle the request, which will send a query to the database. The system will check if there is

an existing SOM model and if yes, it will get the related cluster based on the specific Tweet and display it to the user. If there is no existing model, it will notify the user that there is no SOM model found.

View Specific Cluster Data

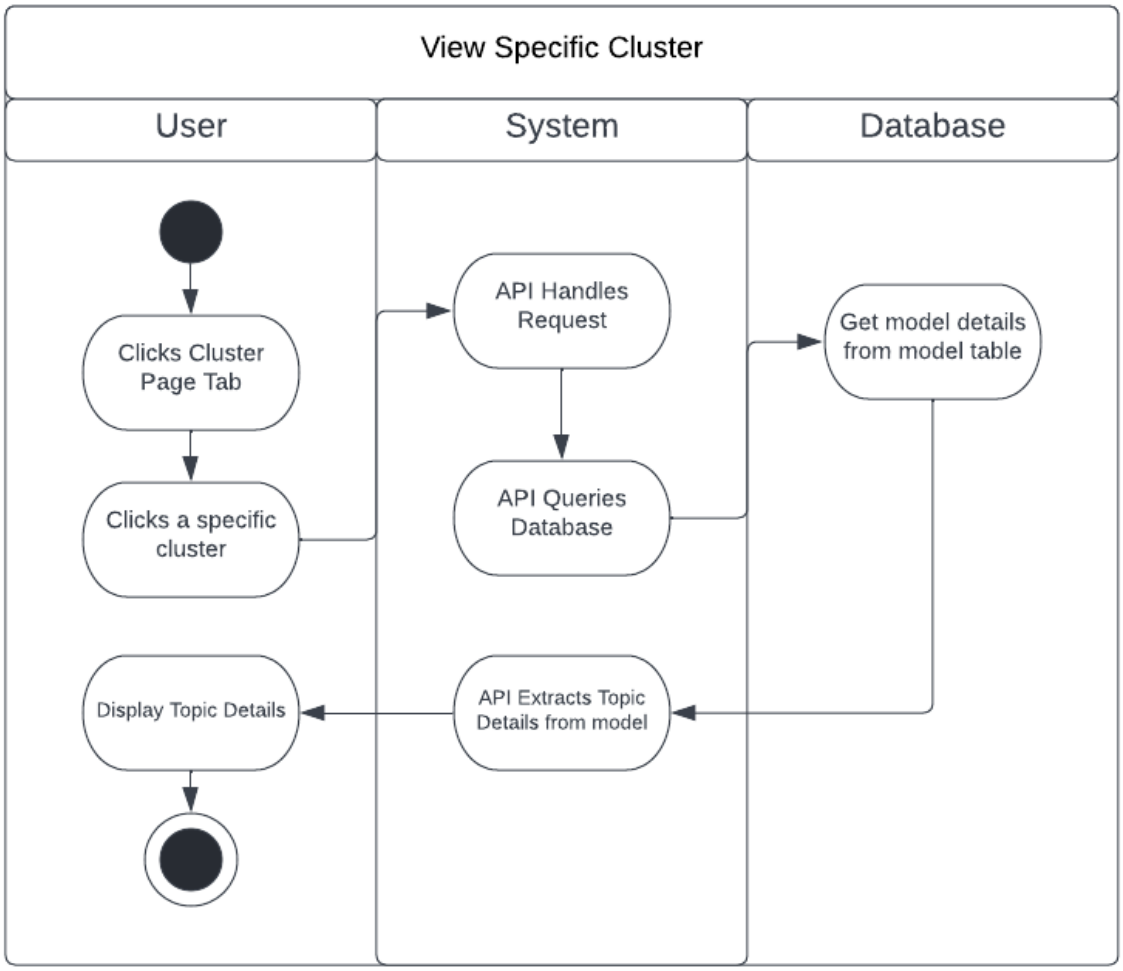


Figure 20: View Specific Cluster Data Activity Diagram

Figure 20 shows the View Specific Cluster Data diagram. Once the user clicks on the cluster page tab then click a specific cluster from the list. The system will send an API request to handle the queries once received. Then the API will get the model details from the collection to display to the user.

View Sentiment of Document

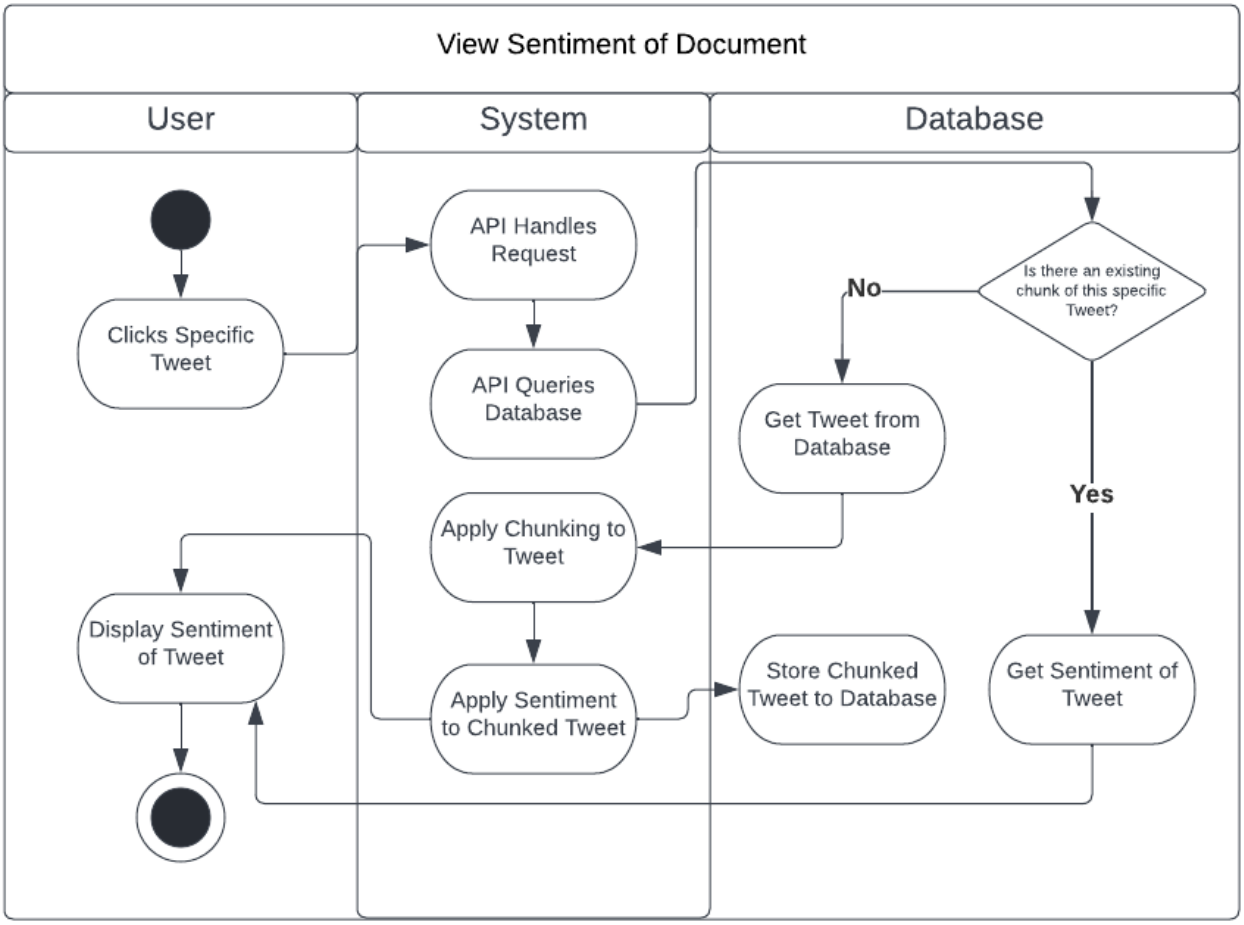


Figure 21: View Sentiment of Document Activity Diagram

Figure 21 shows the View Sentiment of the Document diagram. This diagram is a continuation of Figure 18, which is View Specific Tweet. Once the user clicks on a specific Tweet, it will send a request to the API, which will handle it and query the database. It will check if there is an existing chunk of the specific Tweet; if not, it will get that Tweet from the database and apply both the chunking and sentiment, then store the chunked Tweet in the database. If there is an existing chunk in the database. it will automatically get the sentiment of the Tweet and then display it to the user.

Entity Relationship Diagram

The database used in the system will be served by MongoDB. MongoDB is a NoSQL database and stores data in the form of objects. The data scraped from Twitter uses Javascript Object Notation (JSON); MongoDB is the most compatible way of storing the data.

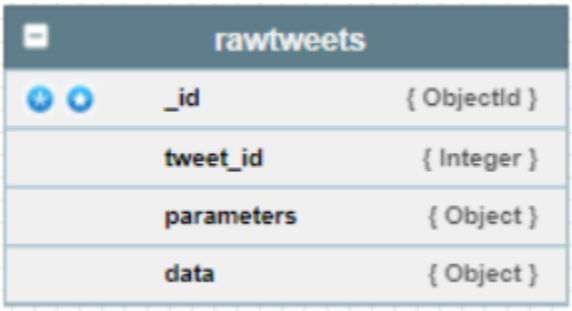


Figure 22: rawtweets Collection

Figure 22 shows the collection of raw tweets that takes the results straight from the scraper. The tweet_id is the unique key of the Tweet; parameters are a group of objects that lists the scraper parameters such as the date of scraping, the queries of search, location, and language. The data contains the complete details of the Tweet.

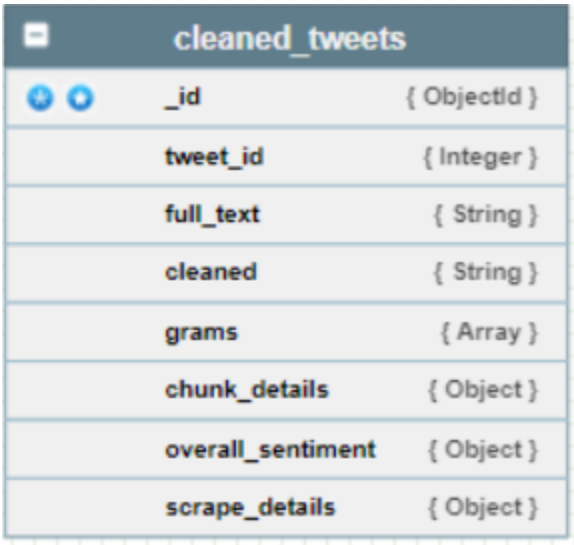


Figure 23: cleaned_tweets Collection

Figure 23 shows the collection of cleaned_tweets which is taken from the results of the pre-processing and cleaning. The tweet_id from the raw tweets table is still used in conjunction with the cleaned tweet, the array or grams, the chunk details from the chunker, and overall_sentiment, where the sentiment score is now applied on the Tweet, and then the scrape_details is taken from the parameters on the raw tweets table.

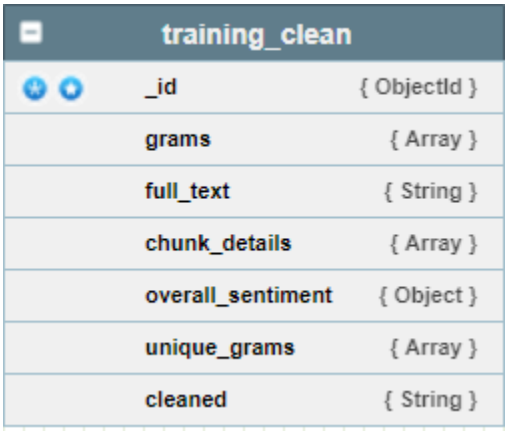


Figure 24: training_clean Collection

Figure 24 shows the grams, the raw tweet, in full_text, with their chunk_details, the overall sentiment, which contains the sentiment of the tweet, the unique grams, and the cleaned tweet. The data is coming from the training set of the SOM model. The method is used once and not rerun in the case of model training.

Figure 25: training_features Collection

Figure 25 shows the collection for the clusters which contain the topic under the field "name." The IDF for the specific word will be used to get the clusters and their relevance.

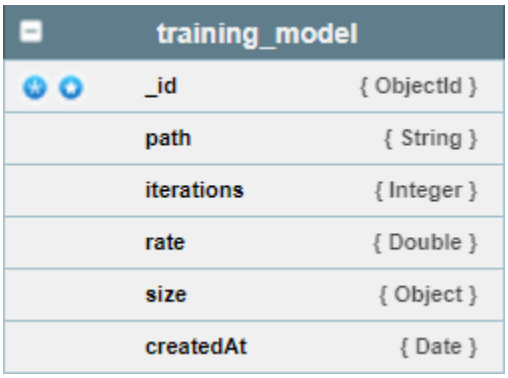


Figure 26: training_model Collection

Figure 26 shows the collection for the SOM model itself. The SOM is saved in a file path locally with its hyperparameters, namely the iterations, learning rate, lattice size, and what date the model is created.

User Interface

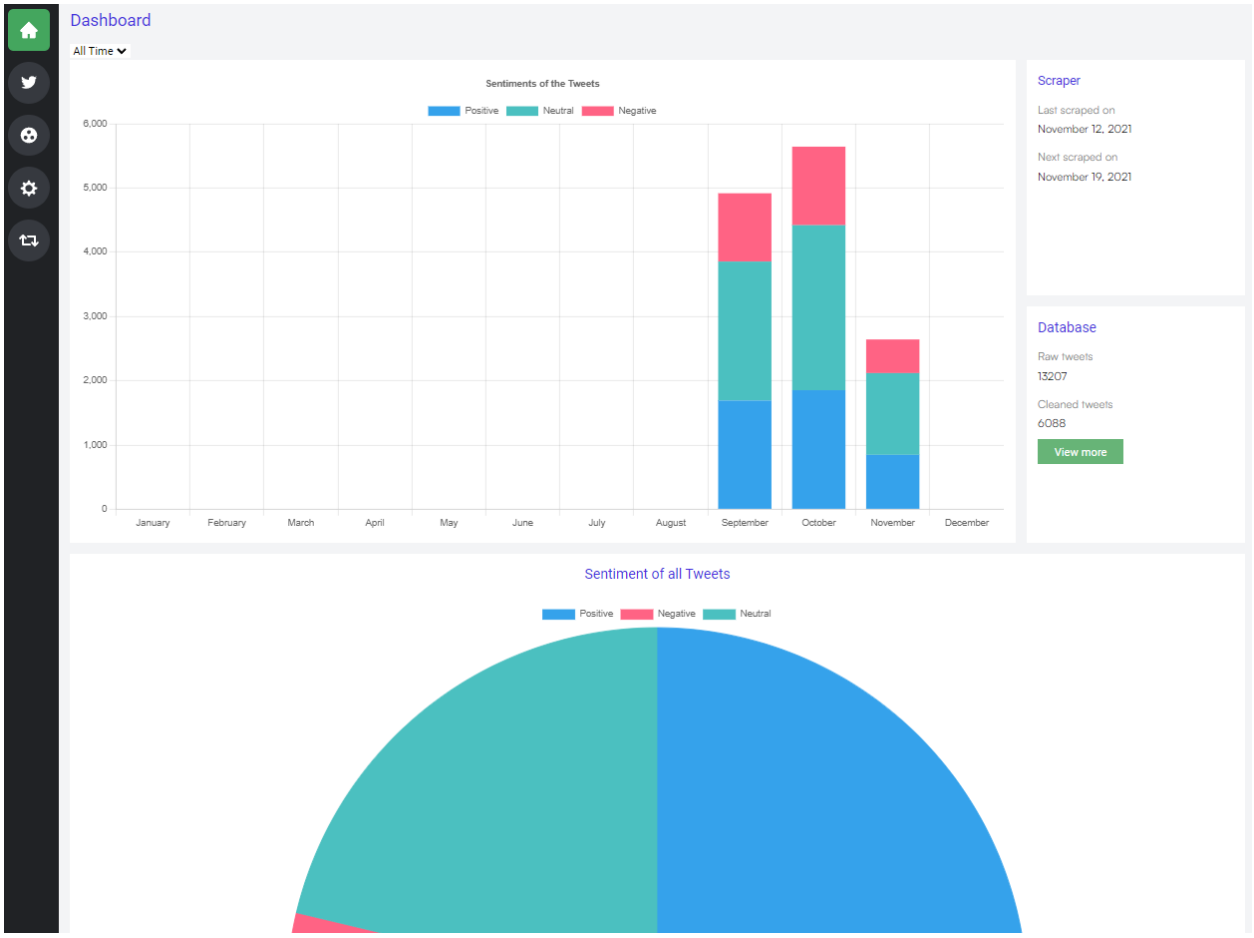


Figure 27: Landing Page

The figure above shows the landing page for Minerva; it contains detailed charts of the scraped data with a stacked bar graph for each sentiment score, namely positive, neutral and negative. The scraped dates are shown with the last scraped date and the schedule for the next scrape date. The user interface also shows the clusters and their sentiments. A pie chart with details on the share of sentiments of tweets is also shown.

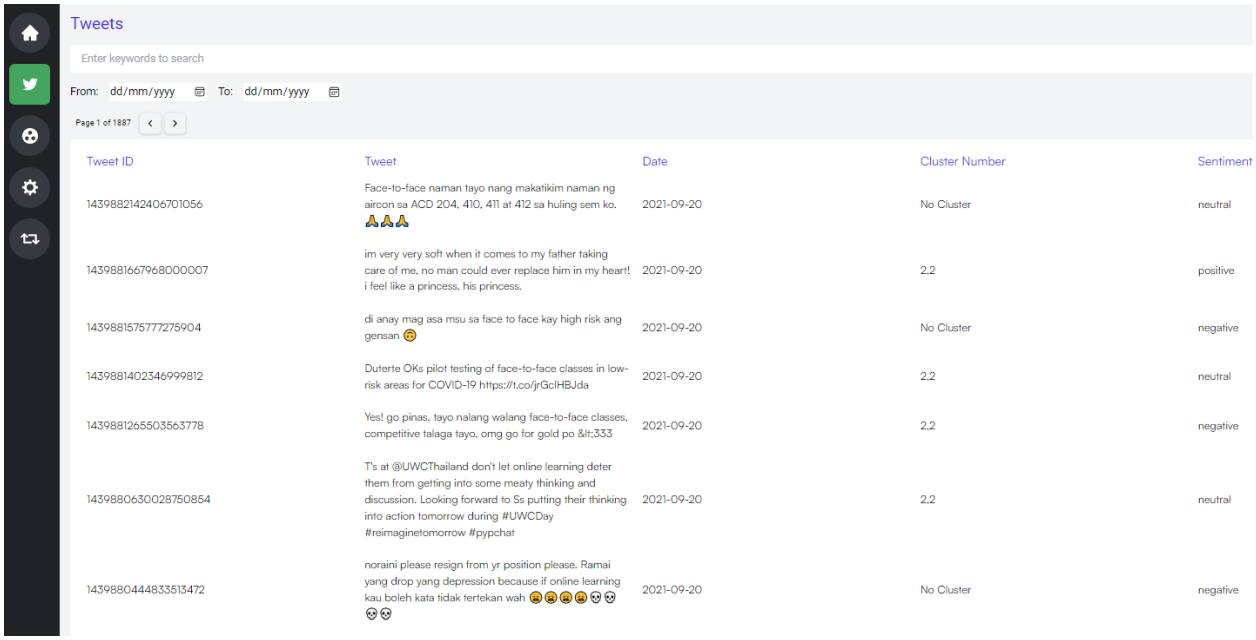


Figure 28: Tweet Page

The figure above shows the complete list of all tweets stored in the database with full details on their tweet id, date tweeted, cluster it belongs to, and the sentiment. A search for keywords and hashtags with a date range is also shown as user input.

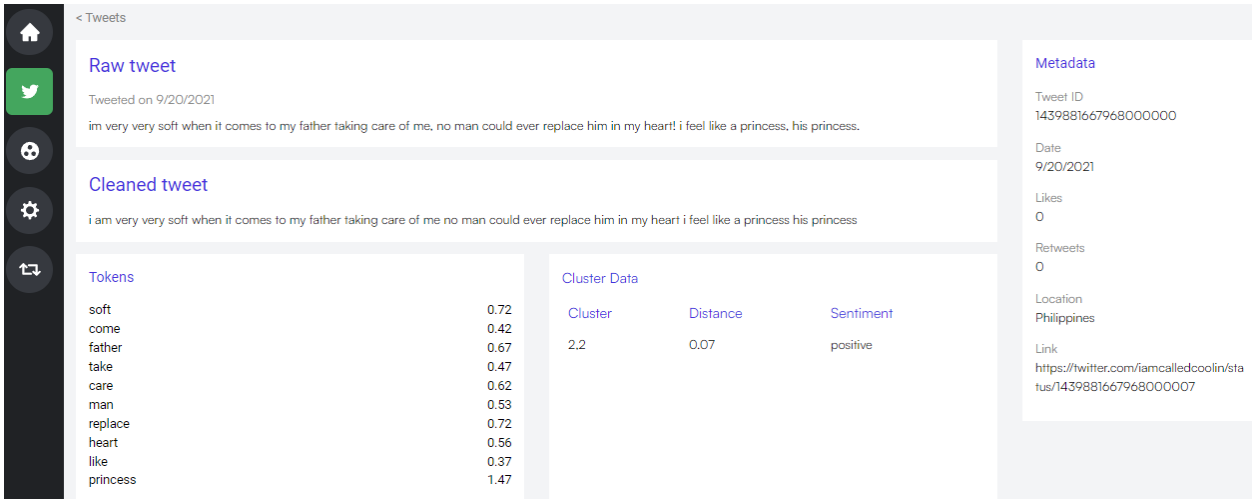


Figure 29: Specific Tweet Page

The figure above shows the full details of a specific tweet when clicked. The interface is a continuation of the tweet page. The data shown are the cleaned tweet, tokens, and corresponding TF - IDF data. Cluster data is also shown with the tweet's location in the SOM model according to its clusters, its Euclidean distance from itself to the best matching unit, and the sentiment of the tweet. Metadata is also shown with the tweet id, unique to the tweet, the date the tweet was created, likes, retweets, its location, and the link to the tweet itself.

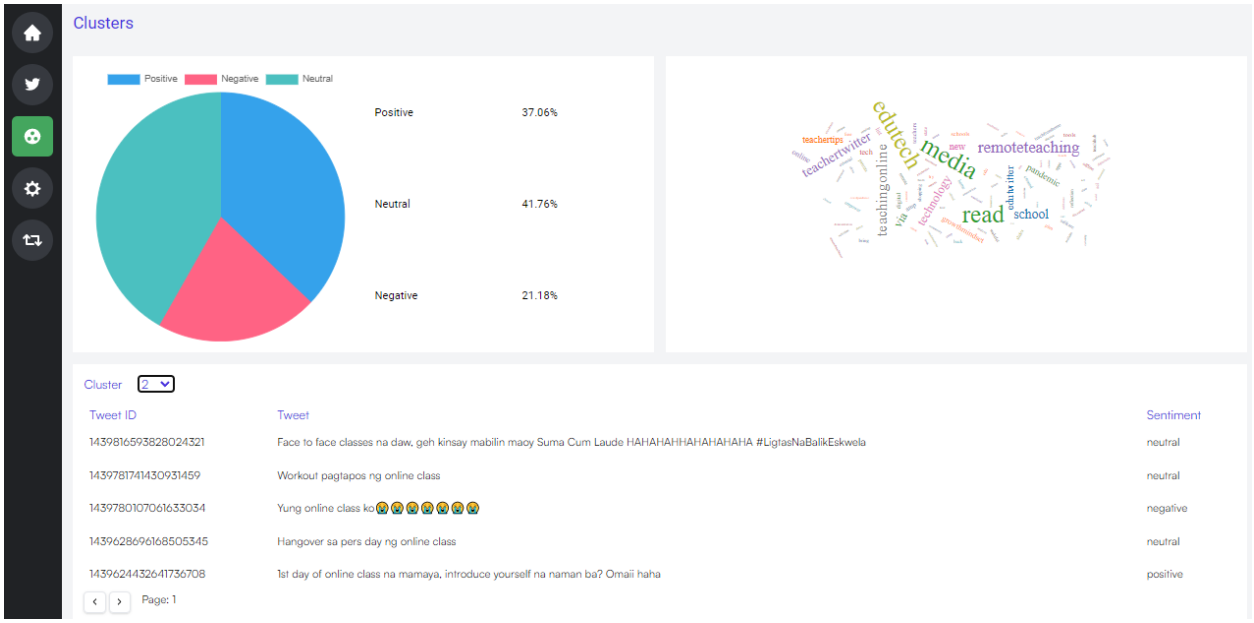


Figure 30: Cluster Page

The figure above shows the cluster page, where the researchers would find the aggregated sentiment of the specific cluster, shown in the form of a pie chart. The word cloud contains the words most frequent in the cluster. A dropdown of the list of clusters can be chosen from the list of tweets belonging to the cluster and its sentiment.

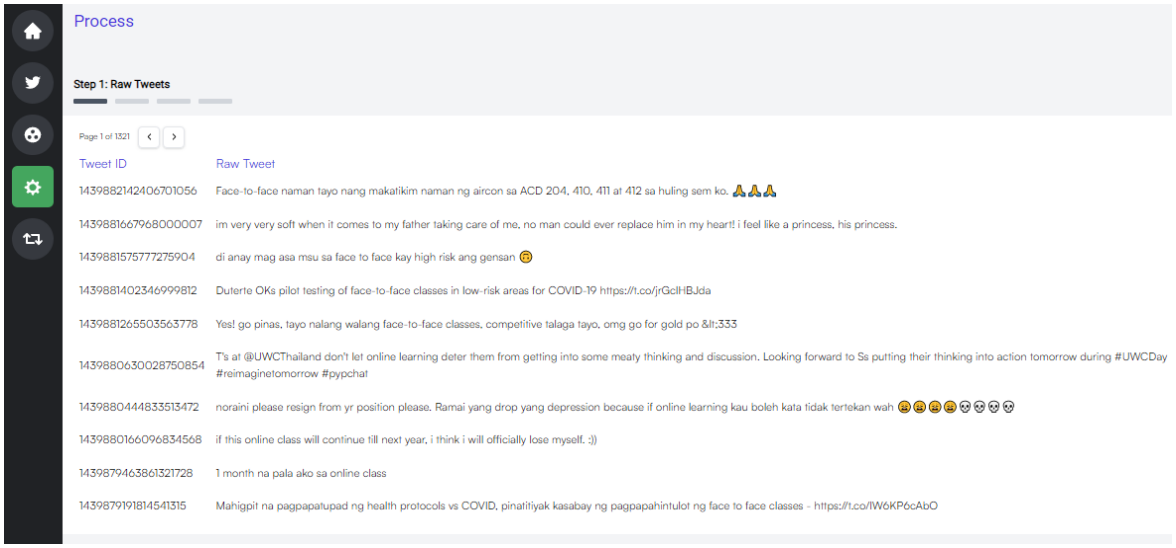


Figure 31: Step by Step Page (Raw Tweets)

The figure above shows the process page. The steps from raw tweets leading to the SOM model. This page contains the visualizations of the step-by-step process of the tweets.

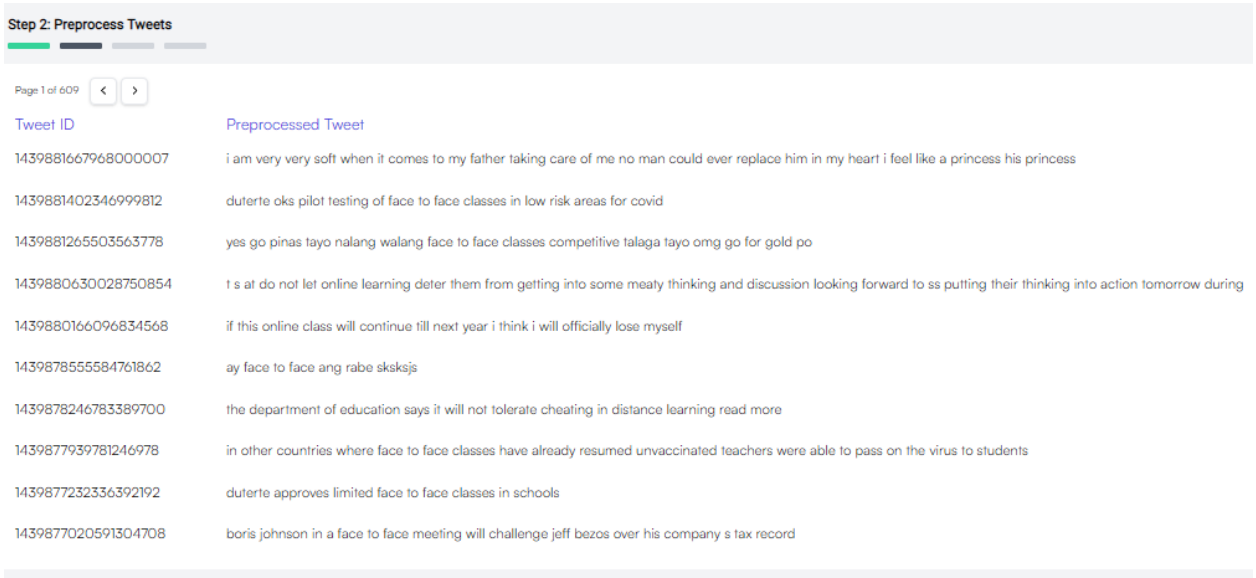


Figure 32: Step by Step Page (Step 2: Preprocess Tweets)

Step 2 contains the already processed tweets. The count of the tweets has lessened since all of the raw tweet data was pruned according to the English language.

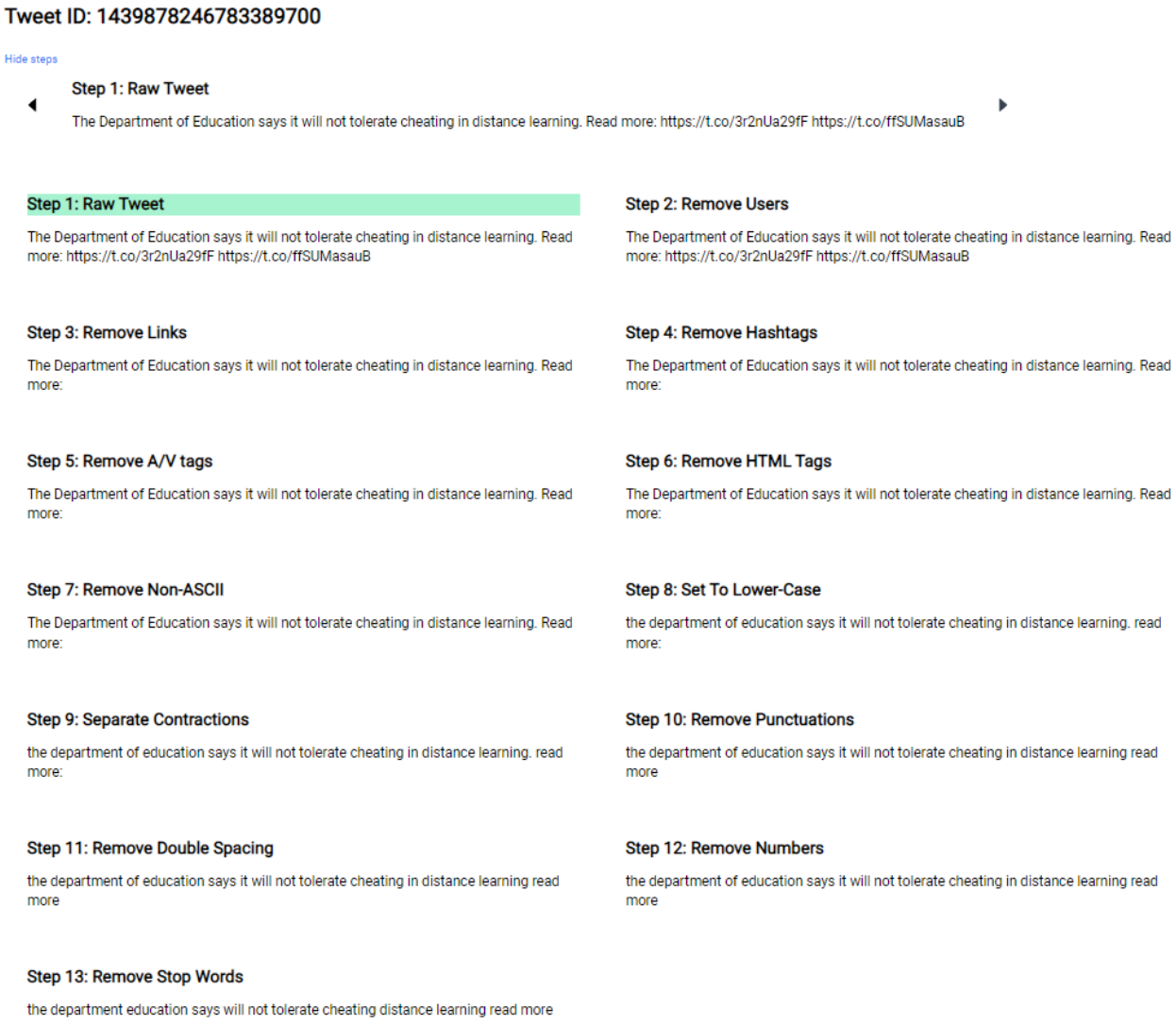
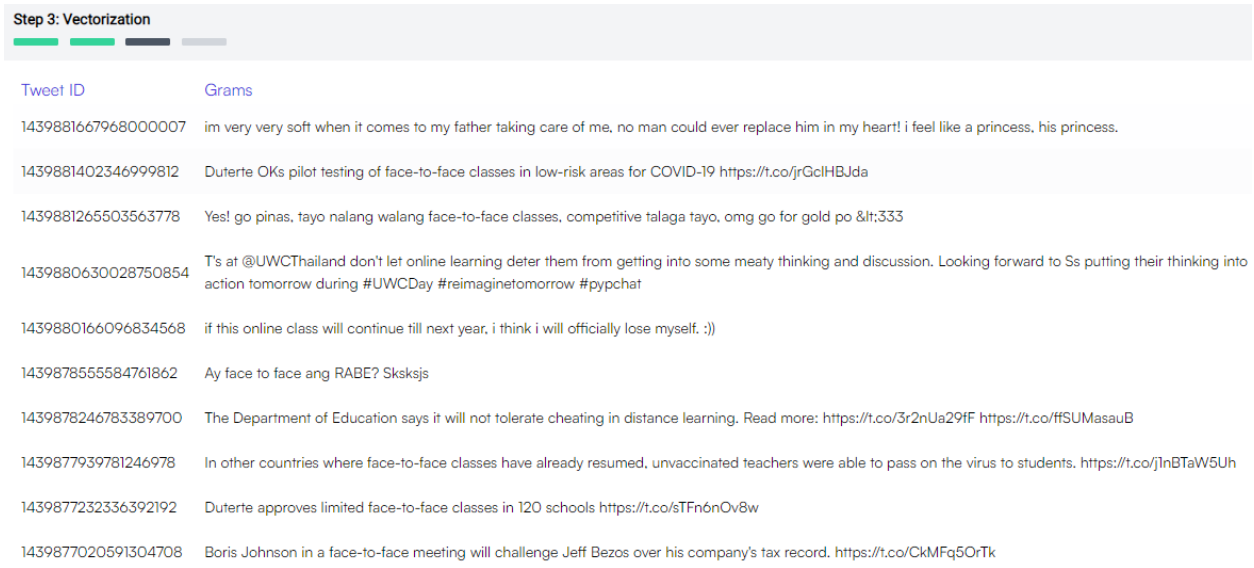


Figure 33: Step by Step Page (Step 2: Preprocess Tweets Full Process)

Once a single tweet has been clicked on Step 2 (Preprocess Tweet) part of the page. An expanded view of the entire tweet pre-processing and cleaning. The entire process of the tweets and their results are shown.

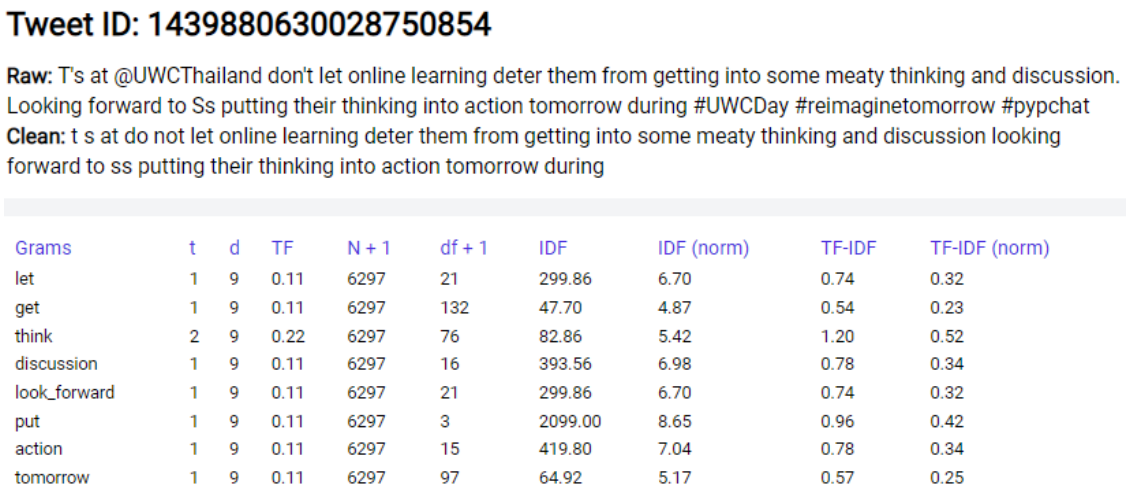


Step 3: Vectorization

Tweet ID	Grams
1439881667968000007	im very very soft when it comes to my father taking care of me, no man could ever replace him in my heart! i feel like a princess, his princess.
1439881402346999812	Duterte OKs pilot testing of face-to-face classes in low-risk areas for COVID-19 https://t.co/jrGclHBJda
1439881265503563778	Yes! go pinas, tayo nalang walang face-to-face classes, competitive talaga tayo, omg go for gold po <333
1439880630028750854	T's at @UWCThailand don't let online learning deter them from getting into some meaty thinking and discussion. Looking forward to Ss putting their thinking into action tomorrow during #UWCDay #reimaginetomorrow #pypchat
1439880166096834568	if this online class will continue till next year, i think i will officially lose myself. :))
1439878555584761862	Ay face to face ang RABE? Sksksjs
1439878246783389700	The Department of Education says it will not tolerate cheating in distance learning. Read more: https://t.co/3r2nUa29fF https://t.co/ffSUMasauB
1439877939781246978	In other countries where face-to-face classes have already resumed, unvaccinated teachers were able to pass on the virus to students. https://t.co/j1nBTaW5Uh
1439877232336392192	Duterte approves limited face-to-face classes in 120 schools https://t.co/sTFn6nOv8w
1439877020591304708	Boris Johnson in a face-to-face meeting will challenge Jeff Bezos over his company's tax record. https://t.co/CkMFq5OrTk

Figure 34: Step by Step Page (Step 3: Vectorization)

Step 3 shows the grams of the tweet. This will be further expanded once a tweet from the list is clicked when the process of the TF - IDF algorithm is shown.



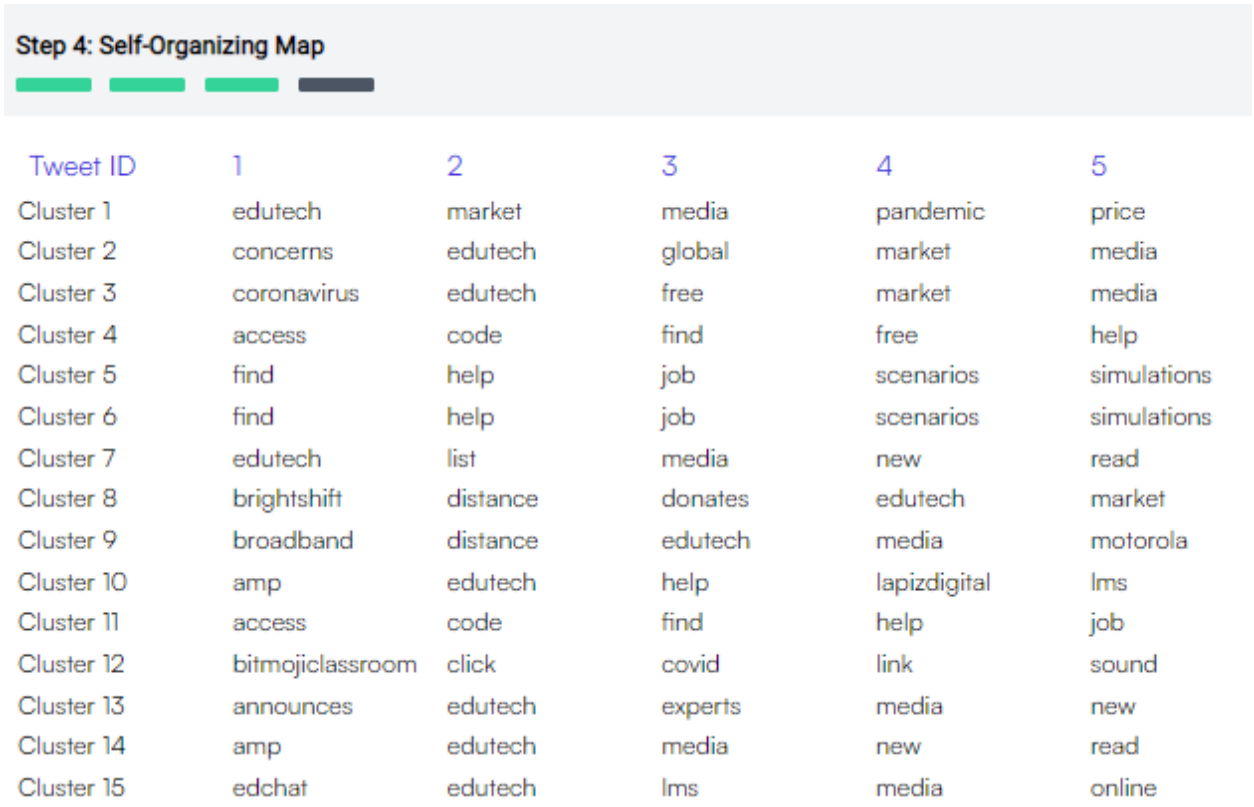
Tweet ID: 1439880630028750854

Raw: T's at @UWCThailand don't let online learning deter them from getting into some meaty thinking and discussion. Looking forward to Ss putting their thinking into action tomorrow during #UWCDay #reimaginetomorrow #pypchat
Clean: t s at do not let online learning deter them from getting into some meaty thinking and discussion looking forward to ss putting their thinking into action tomorrow during

Grams	t	d	TF	N + 1	df + 1	IDF	IDF (norm)	TF-IDF	TF-IDF (norm)
let	1	9	0.11	6297	21	299.86	6.70	0.74	0.32
get	1	9	0.11	6297	132	47.70	4.87	0.54	0.23
think	2	9	0.22	6297	76	82.86	5.42	1.20	0.52
discussion	1	9	0.11	6297	16	393.56	6.98	0.78	0.34
look_forward	1	9	0.11	6297	21	299.86	6.70	0.74	0.32
put	1	9	0.11	6297	3	2099.00	8.65	0.96	0.42
action	1	9	0.11	6297	15	419.80	7.04	0.78	0.34
tomorrow	1	9	0.11	6297	97	64.92	5.17	0.57	0.25

Figure 35: Step by Step Page (Step 3: Vectorization Full Process)

Once a specific tweet has been clicked from the list of Step 3 (Vectorization). The entire process of calculations and their results that lead to the normalized TF - IDF will be shown.



Step 4: Self-Organizing Map

Tweet ID	1	2	3	4	5
Cluster 1	edutech	market	media	pandemic	price
Cluster 2	concerns	edutech	global	market	media
Cluster 3	coronavirus	edutech	free	market	media
Cluster 4	access	code	find	free	help
Cluster 5	find	help	job	scenarios	simulations
Cluster 6	find	help	job	scenarios	simulations
Cluster 7	edutech	list	media	new	read
Cluster 8	brightshift	distance	donates	edutech	market
Cluster 9	broadband	distance	edutech	media	motorola
Cluster 10	amp	edutech	help	lapizdigital	lms
Cluster 11	access	code	find	help	job
Cluster 12	bitmojiclassroom	click	covid	link	sound
Cluster 13	announces	edutech	experts	media	new
Cluster 14	amp	edutech	media	new	read
Cluster 15	edchat	edutech	lms	media	online

Figure 36: Step by Step Page (Step 4: SOM Clusters)

Step 4 shows the summary of the SOM model and its corresponding clusters. All of the clusters will be shown and their top words.

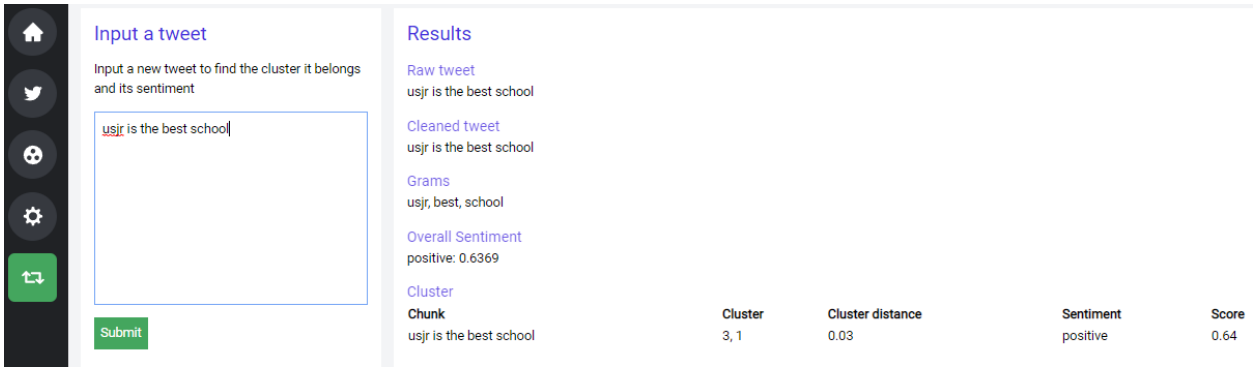


Figure 37: Input Tweet Page

The input tweet page allows the user to input a tweet, which will be analyzed to the SOM model. It also shows the process tweet undergoes before going to the SOM model. The corresponding sentiment analysis of the tweet is also shown. The cluster the tweet belongs to, and its euclidean distance to the best matching unit is also shown.

CHAPTER 3

SOFTWARE DEVELOPMENT AND TESTING

Development Software Platform, Development Environments, and Tools

The key parts of the system are as follows: The Topic Modeler, The Database, The Middleware, and The Web App. A Self-Organizing Map, is used to process the raw tweets to generate topics and sentiments for the data. The database is where the necessary data is stored for the system. The middleware is used to allow the communication of data between the backend and the frontend. Lastly, the web application will be used to view the clusters and sentiments generated by the algorithm.

The Topic Modeler utilizes Python at its core. Python is a popular high-level programming language used to process large amounts of data. It is used in many machine learning and data analysis applications due to the language's community support, readability, and efficacy in processing data. Python also comes with many

valuable tools that will be used in this project. These tools include NumPy, PyMongo, and Twint.

NumPy is a built-in Python library focused on the processing of multidimensional array objects. This tool is vital to the project as the system constantly deals with multidimensional arrays. For example, Self-Organizing maps use an NxM lattice (where N and M are the dimensions representing the number of nodes in the lattice) where each node also contains an array of features with their corresponding values. The project uses NumPy's vast array of methods to help data.

PyMongo is another package aimed at making Python work with MongoDB. It offers the capability to fetch data from a MongoDB-based database. The project uses PyMongo to be able to fetch the scraped tweets from the backend.

Twint is a Python package that provides the ability to scrape tweets. Twint scrapes tweets from Twitter profiles without using Twitter's official API. The package also allows you to scrape tweets with additional parameters. For example, you can set the tweet's location, scrape through keywords, and, more importantly, save the tweets in the database through PyMongo.

MongoDB is the database used by the system. It is a NoSQL database that stores objects inside a table rather than relational tables. Through the previously mentioned packages, the system will be able to scrape tweets and store other vital data.

ReactJS, a web app framework based on Javascript, was used for the front end. ReactJS is a well-supported framework with many packages to deliver a beautiful user interface. In the project, Chart.js was used to display the Pie graphs of the sentiments found in the project.

Flask is a micro web framework tool written in Python. Flask can be used to create a REST API that allows communication between a React client and a MongoDB database.

DEVELOPMENT PROCESS

Web Scraper

For the study, Minerva will be scraping tweets periodically to be later clustered into different topics. It is important to note that these tweets will be used not as training data but only as results for the topic modeling. In this case, this research will be using scraped tweets from Twitter.

Minerva uses a scraping package called Twint in the scraping process. This scraper is run weekly and can get about one to three thousand tweets every run. There are different parameters that the scraper accepts, such as the search keyword and location for the tweet. These parameters define what will be returned by the package in the results.

For these search keywords, the researchers used some common keywords about e-learning in general.

```
DEFAULT_SEARCH_STRING = '"online classes" OR "online class" OR "e-class" OR "online
learning" OR "eclass" OR "face to face" OR "face-to-face" OR "lms" OR "distance
learning" OR "online education" -filter:replies'
```

Figure 38: Search keywords

These keywords are retrieved by scraping initially with a small set of keywords. Afterward, the most frequent keywords from the initial result will be added to form a bigger set of keywords. These keywords, during scraping, will be compared for an exact match by Twitter, so those that do not contain the phrase will not be returned in the results. After preparing the keywords, it can proceed to the scraping process together with the other parameters.

```
def scrape(self, date):
    c = twint.Config()

    c.near = 'Philippines'
    c.Search = self.SEARCH_STRING

    # non-essential parameters
    c.Until = date
    c.Count = True
    c.Filter_retweets = True
    c.stats = True
    c.Store_json = True
    c.Output = date + '.json'

    twint.run.Search(c)
```

Figure 39: All parameters

Using Twint, the researchers defined the important parameters and configured the package according to the needs of the project. As shown in the figure above, there is a near parameter, which is the desired location of the tweet; an until parameter, which is to define at what dates should the package get the tweets from; the search parameter, which is for the keywords to look for; and the filter_retweets to eliminate redundancy in tweets.

Figure 40: Screenshot of the Scraper Results

The figure above shows the result after running the scraper. The essential parts are the *id, tweet, language, created_at,* and *near* properties. These values will be used further into the application such as in the UI and in the SOM process. These values also reflect the parameters that are set before scraping.

After scraping, the raw data will be saved in the database. It will undergo specific changes in its schema. Some values will be added, such as information on how it is scraped. Afterward, it can now proceed to the rest of the steps ahead.

```
def save_to_db(self, date, path):
    file = open(path, encoding="utf-8")
    data = json.load(file)

    table = self.connection['minerva_raw_tweets']['rawtweets']
    table2 = self.connection['minerva_raw_tweets']['cleaned_tweets']

    # save to raw

    for a in data:
        a['full_text'] = a['tweet']

        insert = {
            "scrape_date": date,
            "tweet_id": a['id'],
            "data": a,
            "parameters": {
                "until": date,
                "location": "Philippines",
                "lang": "",
                "query": self.SEARCH_STRING
            }
        }

        table.replace_one({ 'tweet_id': a['id'] }, insert, upsert=True)
```

Figure 41: Saving the raw tweets in the database

```
You, 4 days ago • new module scraper …
for a in data:
    lang = a['data']['language']

    if not lang:
        lang = detect(a['data']['full_text'])


    if lang == 'en':
        text = a['data']['full_text']
        processed = basic_clean(text)
        grams = gram_sentence(processed)

        chunks = chunker(processed)

        chunk_details = []

        for chunk in chunks:
            res = {}
            score = get_sentiment(chunk)
            res['score'] = score
            res['chunk'] = chunk

            chunk_details.append(res)


        insert = {···

        table2.replace_one({ "tweet_id": a['data']['id'] }, insert, upsert=True)
```

Figure 42: Saving the cleaned and filtered tweets in the database

Data Preprocessing

After scraping, the data is then pre-processed to be easier for the algorithm to train. Processing tweets require multiple steps to be fully pre-processed. For the following functions, the researchers used the Python package named RegEX. RegEx is a package used to capture specific substrings in text.

Step 1: Raw Tweet

T's at @UWCThailand don't let online learning deter them from getting into some meaty thinking and discussion. Looking forward to Ss putting their thinking into action tomorrow during #UWCDay #reimaginetomorrow #pypchat

Figure 43: Raw Tweet

```
def remove_users(tweet):
    """Takes a string and removes retweet and @user information"""
    tweet = re.sub('(RT\s@[A-Za-z]+[A-Za-z0-9-_]+)',
                   '', tweet)  # remove re-tweet
    tweet = re.sub('(@[A-Za-z]+[A-Za-z0-9-_]+)',
                   '', tweet)  # remove tweeted at
    return tweet
```

Figure 44: Remove Users

Step 2: Remove Users

T's at don't let online learning deter them from getting into some meaty thinking and discussion. Looking forward to Ss putting their thinking into action tomorrow during #UWCDay #reimaginetomorrow #pypchat

Figure 45: Tweet After Removing Users

For the first step shown in Figure 44, the researchers will first remove the users or under the RegEx with @ followed by a word or letter since it is not needed for the SOM model.

```python
def remove_links(tweet):
    """Takes a string and removes web links from it"""
    tweet = re.sub(r'http\S+', '', tweet)   # remove http links
    tweet = re.sub(r'bit.ly/\S+', '', tweet)  # remove bitly links
    # remove [links]
    tweet = re.sub(r'pic.twitter\S+', '', tweet)
    return tweet
```

Figure 46: Remove Links

Step 3: Remove Links

T's at don't let online learning deter them from getting into some meaty thinking and discussion. Looking forward to Ss putting their thinking into action tomorrow during #UWCDay #reimaginetomorrow #pypchat

Figure 47: Tweet After Removing Links

The next step is the removal of links, as shown in Figure 46. It takes in the Tweet, and takes into account the HTTP links and the URL shorteners. Lastly, the links to pictures are removed from the tweet. Since the sample tweet shown in Figure 47 contains no links before being fed into the remove_links method, the tweet will remain as is.

```python
def remove_hashtags(tweet):
    """Takes a string and removes any hash tags"""
    tweet = re.sub('(#[A-Za-z]+[A-Za-z0-9-_]+)', '', tweet)  # remove hash tags
    return tweet
```

Figure 47: Remove Hashtags

Step 4: Remove Hashtags

T's at don't let online learning deter them from getting into some meaty thinking and discussion. Looking forward to Ss putting their thinking into action tomorrow during

Figure 48: Tweet After Hashtags Removed

The next step is the removal of hashtags, as shown in Figure 47. It takes in the tweet and then is removed by RegEx if there is a certain # followed by a word or letter.

```
def remove_av(tweet):
    """Takes a string and removes AUDIO/VIDEO tags or labels"""
    tweet = re.sub('VIDEO:', '', tweet)  # remove 'VIDEO:' from start of tweet
    tweet = re.sub('AUDIO:', '', tweet)  # remove 'AUDIO:' from start of tweet
    return tweet
```

Figure 48: Remove Audio and Video Labels

Step 5: Remove A/V tags

T's at don't let online learning deter them from getting into some meaty thinking and discussion. Looking forward to Ss putting their thinking into action tomorrow during

Figure 49: Tweet After A/V Tags Removed.

The next step is the removal of audio and video labels found at the start of the tweet, as shown in Figure 48. This will be removed by RegEx under the "VIDEO:" and "AUDIO:." Since the sample processed tweet shown in Figure 49 does not contain an Audio and Video label, the tweet will remain as is

```
def remove_html_entities(string):
    res = re.sub(
        r'&(gt|lt|amp|nbsp|quot|apos|cent|pound|yen|euro|copy|reg);', '', string)
    return res
```

Figure 50: Remove HTML Entities

Step 6: Remove HTML Tags

T's at don't let online learning deter them from getting into some meaty thinking and discussion. Looking forward to Ss putting their thinking into action tomorrow during

Figure 51: Tweet After HTML Tags Removed.

The next step is found in Figure 50, where the researchers will remove the HTML entities such as " " for non-breaking space, "&" for the ampersand, """ and for double quotation marks. This will be catered by RegEx, which takes in a specific tweet.

```
def remove_non_ascii(string):
    return string.encode("ascii", "ignore").decode()
```

Figure 52: Remove Non - Ascii

Step 7: Remove Non-ASCII

T's at don't let online learning deter them from getting into some meaty thinking and discussion. Looking forward to Ss putting their thinking into action tomorrow during

Figure 53: Tweet After Removed Ascii

Next is to remove ASCII, such as tweets with ASCII art. Since Figure 52 does not show an ASCII value, the tweet will remain as is.

```
def lower(string):
    return str(string).lower()
```

Figure 53: Apply Lowercase

Step 8: Set To Lower-Case

t's at don't let online learning deter them from getting into some meaty thinking and discussion. looking forward to ss putting their thinking into action tomorrow during

Figure 54: Tweet After Applied Set to Lowercase

The next step is to apply lowercase on the tweet, as shown in Figure 53. This function takes in a tweet and then applies lower to the tweet. This is useful for uniformity on the entire tweet.

```
def fix_contractions(string):
    return contractions.fix(string)
```

Figure 55: Separate Contractions

Step 9: Separate Contractions

t's at do not let online learning deter them from getting into some meaty thinking and discussion. looking forward to ss putting their thinking into action tomorrow during

Figure 56: Tweet After Contractions Separated

Contractions are shortened forms of a word in a document such as “I'm”, “Let's“, “It's“, etc. This needs to be expanded to its original form, as shown in Figure 55, to know the true meaning of the tweets. If only the punctuations between the words are removed, this may cause grammatical errors, for example in the contraction it's when removing the punctuation results in "it s" rather than "it is". This creates a separate meaning to the intended tweet. Since the sample tweet in Figure 56 shows no contractions, the tweet will remain as is.

```
def remove_punctuations(string):
    return re.sub('[' + punctuation + ']+', ' ', string)
```

Figure 56: Remove Punctuations

Step 10: Remove Punctuations

t s at do not let online learning deter them from getting into some meaty thinking and discussion looking forward to ss putting their thinking into action tomorrow during

Figure 57: Tweet After Punctuations Removed

The next step is to remove punctuation on the tweets, as shown in Figure 56. This is removed by RegEx and is returned to the system. Since the sample processed tweet shown in Figure 57 does not contain punctuations, the tweet will remain as is

```
def remove_double_spacing(string):
    return re.sub('\s+', ' ', string)
```

Figure 58: Remove Double Spacing

Step 11: Remove Double Spacing

t s at do not let online learning deter them from getting into some meaty thinking and discussion looking forward to ss putting their thinking into action tomorrow during

Figure 59: Tweet After Double Spacing Removed

Next is to remove the double spacing on tweets, as shown in Figure 58. This is needed to clean up the returns on the earlier, steps such as removing some texts for uniformity of data. Since the sample processed tweet shown in Figure 59 does not double spacing, the tweet will remain as is

```
def remove_numbers(string):
    return re.sub('([0-9]+)', '', string)
```

Figure 60: Remove Numbers

Step 12: Remove Numbers

t s at do not let online learning deter them from getting into some meaty thinking and discussion looking forward to ss putting their thinking into action tomorrow during

Figure 61: Tweet After Numbers Removed

Next is shown in Figure 60 for numbers in the tweet since the algorithm only uses the words in the tweet and not the numbers. This is removed by the library RegEx. Since the sample processed tweet shown in Figure 61 does not contain numbers, the tweet will remain as is

```
def clean_token(word):
    if '_' in word:
        split = word.split('_')

        if len(split[0]) > 2 and len(split[1]) > 2 and split[1] not in STOPWORDS and split[0] not in STOPWORDS:
            return p.lemmatize(split[0]) + '_' + p.lemmatize(split[1])

    elif len(word) > 2 and word not in STOPWORDS:
        return p.lemmatize(word)
```

Figure 62: Remove Stop Words

Step 13: Remove Stop Words

not let online learning deter them from getting into some meaty thinking and discussion looking forward putting their thinking into action tomorrow during

Figure 63: Tweet After Stop Words Removed

The last step for preprocessing is to remove the stop words, a list of commonly used words in the English language. This is essential for the system to focus on the crucial texts and eliminate noise in the document.

TF - IDF Vectorization

Tweet ID: 1439880630028750854

Raw: T's at @UWCThailand don't let online learning deter them from getting into some meaty thinking and discussion. Looking forward to Ss putting their thinking into action tomorrow during #UWCDay #reimaginetomorrow #pypchat
Clean: t s at do not let online learning deter them from getting into some meaty thinking and discussion looking forward to ss putting their thinking into action tomorrow during

Grams	t	d	TF	N + 1	df + 1	IDF	IDF (norm)	TF-IDF	TF-IDF (norm)
let	1	9	0.11	6297	21	299.86	6.70	0.74	0.32
get	1	9	0.11	6297	132	47.70	4.87	0.54	0.23
think	2	9	0.22	6297	76	82.86	5.42	1.20	0.52
discussion	1	9	0.11	6297	16	393.56	6.98	0.78	0.34
look_forward	1	9	0.11	6297	21	299.86	6.70	0.74	0.32
put	1	9	0.11	6297	3	2099.00	8.65	0.96	0.42
action	1	9	0.11	6297	15	419.80	7.04	0.78	0.34
tomorrow	1	9	0.11	6297	97	64.92	5.17	0.57	0.25

Figure 64: Sample tweet with TF - IDF computations, results and normalization

Before proceeding to cluster the documents, they should be represented first in a format that the computer must understand. This process of converting a document to this format is called vectorization. For this research, TF IDF vectorization is performed manually on the dataset.

The first step in the process is to prepare the documents for input. In the previous section, the data is pre-processed and tokenized into unigrams, bigrams. Afterward, a bag of words must be generated. A bag of words is a matrix of documents-to-words that contains the number of occurrences a word has in a particular document. To build this matrix, the unique words must be first extracted to be used as columns. Only the most used words will be retained, and the less common ones will be discarded.

```
def bow(doc_grams, max=4000, getIdf = False):
    unique = {}

    for doc in doc_grams:
        for gram in doc:
            if gram not in unique:
                unique[gram] = 1
            else:
                unique[gram] = unique[gram] + 1


    unique = list(
        sorted(unique.items(), key=lambda item: item[1], reverse=True))[:max]

    unique = [val[0] for val in unique]
```

Figure 65: Getting the top 4000 unique words

0	0.5547	0	0	0	0	0	0.27735	0.27735	0.27735	0.27735
0.23552	0	0.30278	0.30278	0	0	0	0	0	0	0
0	0	0	0	0.5547	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0.24246	0	0	0	0	0	0	0	0	0	0
0.34825	0	0.44771	0	0	0.44771	0.44771	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0.25728	0	0	0.33077	0	0.33077	0.33077	0	0	0	0

Figure 66: Normalized TF - IDF Matrix of Unique Words to Tweets

These unique words will be the columns, and the rows would be the Tweets. The TF - IDF will then be normalized by using L2 normalization. This is done by dividing the matrix by the Euclidean distance of all the features in the document.

Self - Organizing Map

The Self-Organizing Map (SOM) is an unsupervised, clustering, machine learning algorithm trained through competitive learning. SOM takes in a collection of pre-processed documents, plots them onto a lattice of features, updates the weights until a certain number of steps, and then matches these documents to the best matching unit. SOM is primarily used to create a low-dimensional representation of a

high-dimensional dataset while maintaining the topological structure of the data. In this project, the main focus will not be on representing the dataset on a 2D plane but on generating a topic model based on the weights of each node in the SOM.

The SOM will take in the full TF-IDF matrix, some hyperparameters, and an optional callback function to work on the matrix. It is worth mentioning that, for the training of the model, the researchers used a different dataset of tweets than the ones scraped by the scraper. However, they still undergo the same pre-processing procedure.

The first step is to initialize the SOM's hyperparameters. The hyperparameters are as follows: *lattice size, learning rate, and epochs/steps.*

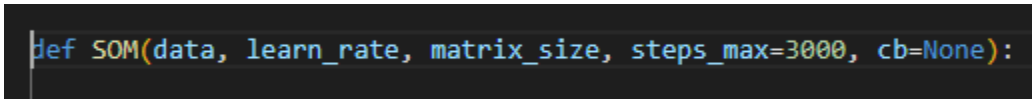


Figure 67 : The SOM function name and its hyperparameters

The first hyperparameter, lattice size, dictates the number of clusters in the SOM. In the project, this also dictates the number of topics the model produces. It is represented by *n x m,* where *n* and *m* is an integer representing rows and columns, respectively. An example of a lattice size would be a *2 x 2* lattice, where it would

produce *4* clusters. To compare this with more traditional topic modeling algorithms, this is equivalent to the *number of topics*.

The second hyperparameter is the learning rate. This hyperparameter directly affects the formula that updates the weights. It is a value ranging from 0 to 1 inclusive. From the name itself, it implies how fast the algorithm will learn. In SOM's case, this means how aggressive it will be in clustering the data.

The third and final hyperparameter is how many epochs the SOM will do. For each epoch, the SOM will update the weights of its lattice depending on the best matching unit of that epoch. This means that more epochs mean that the dataset will also be fitted more. Inversely, the lesser the epochs means that the lattice will not be updated as much.

Once the SOM's hyperparameters have been set, it is time to initialize the lattice. The researchers will do this by initializing a 3-dimensional array with the dimensions of n x m x number of features. The number of features is equivalent to the number of features in the TF-IDF. This is important because each node having the same amount of features as the document's TF-IDF is what allows the documents to be matched with a node. After initializing, the researchers will populate the matrix with a random floating number between 0 and 1 inclusive. That is why it is essential to normalize the TF-IDF

because if the input matrix has values greater than one or less than zero, it will not correctly cluster the dataset. Along with this, we will also initialize the maximum range of the neighborhood. The maximum range is calculated by doing n + m. The maximum range is indicated because the neighborhood is calculated through Manhattan distance. Since n and m are considered the maximum of the rows and columns of the lattice, it makes sense that these will be the values used to get the maximum range of the neighborhood function.

```
# 1 Initialize some constants

(steps, num_features) = data.shape
(row, col) = matrix_size
range_max = row + col

# 2 Create the initial matrix

matrix = np.random.random_sample(size=(row, col, num_features))
```

Figure 68 : Initializing the SOM in the function

Afterward, the researchers will add a *percent* variable that will start from 1 at the beginning of training and decrease to 0 by the end. This percent value will then be multiplied with the *learning rate* and *max range* to decrease their strength in their respective formulas as the training progresses.

```
percent = 1.0 - ((current_step * 1.0) / steps_max)
```

Figure 69 : Getting the percentage

The SOM model then, for each epoch, randomly selects a document within the TF-IDF matrix for training. Afterward, the SOM will iterate through the entire lattice finding the best matching unit concerning the input vector. The best matching unit is defined by the shortest distance between the input vector and all of the nodes. The distance is calculated by using the *euclidean distance formula*. This is done by getting the square root of the sum of the squares of the differences of the vectors between the input and the current node it is evaluating. This means that the normalized TF-IDF values are subtracted by the weights inside a node through matrix subtraction. These values are then squared and summated before getting their square root to get their Euclidean distance. The Euclidean distance formula applies the Pythagorean theorem, where the distance between two points is calculated.

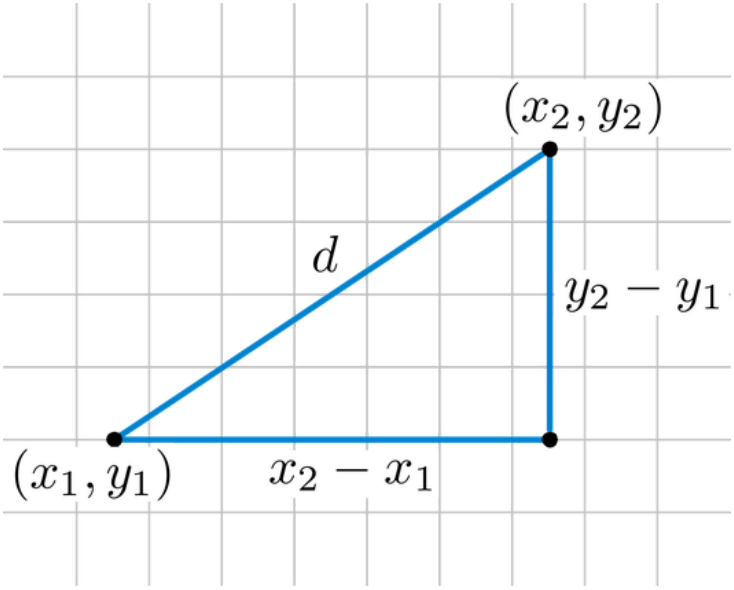


Figure 70 : The Pythagorean Theorem. The basis for the euclidean distance formula

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

Figure 71 : The Euclidean Distance Formula

```
def find_bmu(matrix, input, matrix_size):
    (row, col) = matrix_size

    num_features = len(input)

    bmu = (0, 0)
    nearest = 10000000

    for i in range(row):
        for j in range(col):

            distance = euc_distance(input, matrix[i][j], num_features)

            if distance < nearest:
                nearest = distance
                bmu = (i, j)
```

Figure 72: find_bmu is the function used to find the best matching unit for the input in the SOM

```
def euc_distance(input, cell, num_features):
    sum = 0

    for n in range(num_features):
        sum = sum + pow(input[n] - cell[n], 2)

    return sqrt(sum)
```

Figure 73: euc_distance is the function used to find the euclidean distance

After finding the best matching unit, the node weights and its surrounding nodes are updated. The nodes to be updated are determined by their Manhattan distance toward the best matching node. If its distance is within a specific range, then the weights are allowed to be updated; otherwise, it is left unchanged. This distance, range,

and the learning rate at which the nodes are updated have their corresponding formula. To find the distance, the first step is subtracting the node's row number from the BMU's row number; the second, subtracting the node's column number from the BMU's column number; and lastly, adding the absolute values of both differences to calculate the distance.

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Figure 74 : The Manhattan Distance formula

```
def man_distance(bmu_row, bmu_col, row, col):
    return np.abs(bmu_row-row) + np.abs(bmu_col-col)
```

Figure 75: man_distance is the function used to find the manhattan distance of the input and node

After getting this information, the distance will be checked to see whether it is within a specific range. This range is calculated by taking the total number of iterations multiplied by the percent of iterations left. This means that the range decreases as the number of iterations increases. The number of nodes to be updated will also decrease.

The learning rate is also updated as the number of iterations increases. Like the range, the rate is calculated by taking the original learning rate value and multiplying it by the percent of iterations left.

The calculated distance, range, and learning rate will then be used to update the weights surrounding the BMU. If the distance of a node is within this specific range, a formula is then applied:

$$w_{i,j}\,(s+1) = w_{i,j}\,(s) + cl[x(s) - w_{i,j}\,(s)]$$

Figure 76: The formula to update the weights of a node

```
# Weight Updating (w/ man_distance)

if man_distance(bmu_row, bmu_col, i, j) < curr_range:
    matrix[i][j] = cell + curr_rate * (input-cell)
```

Figure 77: Using the distance and the range to determine which to update

The algorithm loops through the lattice and checks if the current cell is eligible for an update. If the condition is proper, it applies the formula above. The new weight is represented by $w_{i,j}\,(s+1)$, where *w* is the lattice, *i* the row, *j* the column, and *s* the current iteration. The formula then takes the current weight $w_{i,j}\,(s)$ and adds the product

of the learning rate *cl* and the matrix difference of the best matching unit *x(s)* and the current weight $w_{i,\ j}$ *(s)*. The result is then assigned to the cell to represent the new weights for the next iteration.

Once the SOM has been trained, the algorithm can proceed to cluster. In the project, the data being clustered is the data that has been scraped.

Document Chunking

The documents are chunked using a pre-trained chunker. This chunker is from the NLTK and is pre-trained using an English dataset from NLTK also. The researchers used this approach because the complexity and various syntax of the English language have proved difficult to manage manually.

```
from nltk.corpus import state_union
from nltk.tokenize import PunktSentenceTokenizer

train_text = state_union.raw("2005-GWBush.txt")
tokenizer = PunktSentenceTokenizer(train_text)


def chunker(string):
    return tokenizer.tokenize(string)
```

Figure 78: Chunk function

The chunker works by taking a string with proper punctuation and then splitting it based on the learned pattern by the tokenizer. The chunks processed from this function will be used with the sentiment analysis. Every chunk in a document is independent and will be assigned to its topic. This will mean that a tweet or a document might contain several topics, each with its corresponding sentiment score.

Sentiment Analysis

The sentiment analysis is handled by the VADER sentiment analysis tool. VADER is a lexicon and rule-based sentiment analysis tool specializing in getting the sentiment from social media datasets. The sentiment analysis function used in the application utilizes both the chunker and VADER because each chunk could belong to a different topic and can have a different sentiment. It returns a normalized sentiment score between -1 (extreme negative) to +1 (extreme positive). Chunking will be used with sentiment analysis to get the tweets with contradicting sentiments or meanings. It takes a chunked tweet to separate the tweets, the separated tweet will feed one by one to the sentiment analysis, and then the scores will be averaged, and the resulting score will be used to get the sentiment score.

```python
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()

def get_sentiment(tweet):
    score = analyser.polarity_scores(tweet)['compound']
    if score <= -0.05:
        sentiment = "negative"
    elif score < 0.05:
        sentiment = "neutral"
    elif score > 0.05:
        sentiment = "positive"

    return (sentiment, score)
```

Figure 79: Sentiment analysis function

ACCURACY TESTING

Topic Coherence

Topic coherence will be utilized to optimize the model since the Self-Organizing Map model is used for topic modeling. The researchers used topic coherence with the function UMass since it takes a topic-to-word matrix, which happens to be the output of our SOM model. The topic coherence will be looped for each testing of hyperparameters. A Tweet of (m, n) is the count of Tweets containing the words m and n plus a smoothing factor of 1, then divided by the number of Tweets containing m. It is then calculated for its logarithmic value. The arithmetic mean of all values in the word is then returned.

$$score(v_i, v_j, \epsilon) = \log \frac{D(v_i, v_j) + \epsilon}{D(v_j)}$$

Figure 80: UMass Function

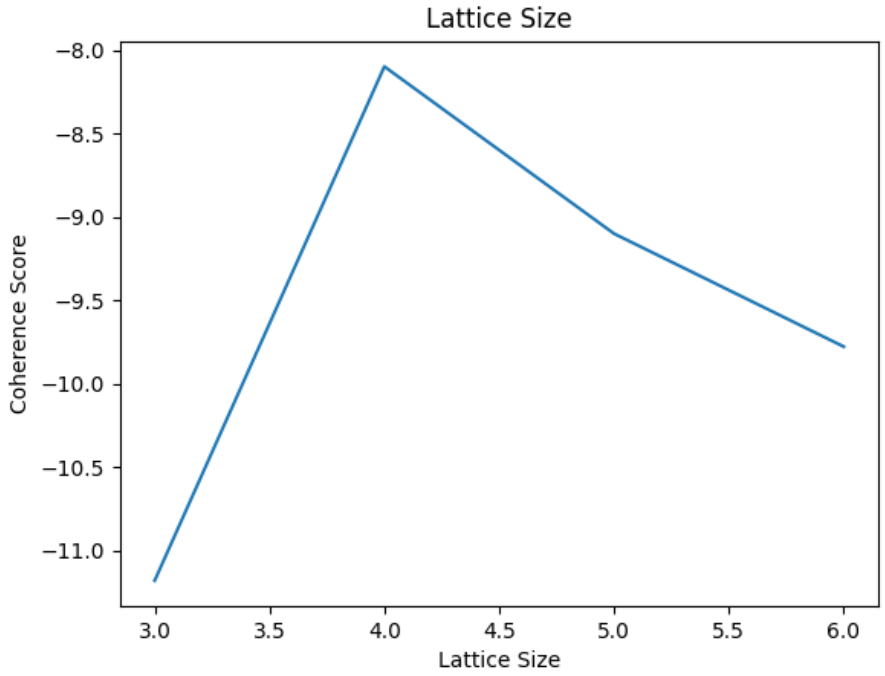


Figure 81: Lattice Numbers

Figure 79 is the graph for testing figures for the lattice size in the SOM matrix, which lets us know the total number of features in the model. Here it is shown that a four by four matrix contains the closest score to 0, where it is the most optimal. The researchers are testing a square matrix for easier visualization, although other shapes would also work. The lattice size determines the number of clusters needed for the model to work with, in this case, 16 clusters.

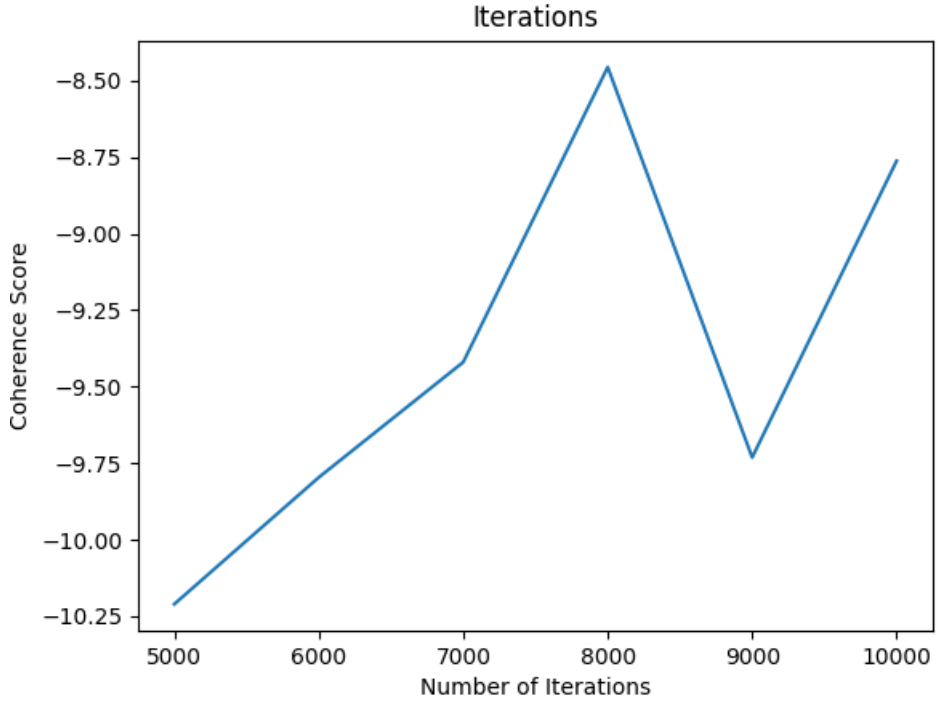


Figure 82: Iterations

After getting the lattice size, the optimal number of iterations will be next for hyperparameter tuning. Here, the researchers see that 8000 gives us the best coherence score in the model. Keep in mind that the right lattice size for the model will be used in testing for the iterations. The iterations are needed to know how many times the model would update the weights, in this case, 8000.

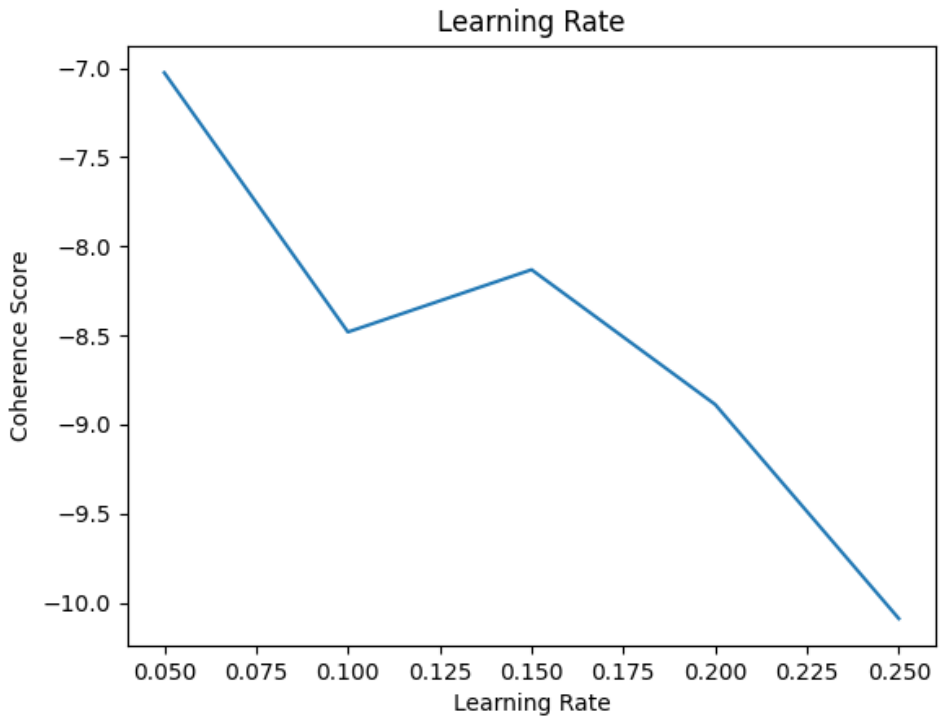


Figure 83: Learning Rate

After getting the number of iterations, the needed learning rate for the model will be used for hyperparameter tuning of the SOM model. The learning rate for SOM will determine how aggressive the SOM will update the number of weights. The output of our topic coherence is 0.05. This, in conjunction with the other tested hyperparameters, will be used to build the final SOM model.

Silhouette Index

Self-Organizing Map is an unsupervised clustering algorithm, so it can be complicated to validate the map results. However, the method can compare the clusters to other clusters on the Map. In this project, the **silhouette index** is used to achieve this.

Silhouette analysis refers to a method of interpretation and validation of consistency within data clusters. The silhouette value measures how similar an object is to its cluster (cohesion) compared to other clusters (separation). It can be used to study the separation distance between the resulting clusters. The silhouette plot displays how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like the number of clusters visually.

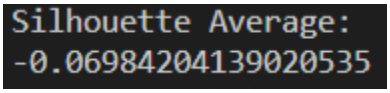


Figure 84: Silhouette Average Score

PERFORMANCE ANALYSIS

Multi - Threading

To speed up the processes involved in the system, the researchers utilized multi-threading. Multi-threading is a technique wherein a computer instruction is run concurrently with another instruction or even by a portion of the same instruction. In this system, the researchers applied this concept to CPU-intensive processes, such as the TF-IDF. It is also important to remember that while this process might not be as complex as the core of the application, the SOM, it still takes up much time, primarily since it is coded manually and the algorithm might be inefficient logically.

```
def tf_idf(document_array, bow=None):
    start_time = time.time()

    if bow is None:
        (bow, x, y) = bow(document_array)

    matrix = np.zeros(bow.shape, dtype=float)
    doc_count = len(document_array)

    col_sums = np.count_nonzero(bow, 0)

    data = []
    matrix = []

    for i, row in enumerate(bow):

        data.append((row, doc_count, col_sums))

    with concurrent.futures.ThreadPoolExecutor(4) as executor:
        for _ in executor.map(_tf_idf_sub, data):
            matrix.append(_)

    return np.asarray(matrix)
```

Figure 85: Using four threads to make the tf_idf matrix

The figure above shows the TF-IDF process, and the multi-threading concept applied. The steps applied are as follows:(1) documents are divided into four subparts; (2) the TF-IDF sub-function is run concurrently on each subpart; (3) the results are concatenated to form a single and final result. Ideally, this technique is expected to run about four times faster than the single-threaded method.

```
def _tf_idf_sub(data):
    (row, doc_count, col_sums) = data
    total_distance = 0
    res = np.zeros(len(row), dtype=float)

    word_count = np.sum(row)

    for j, col in enumerate(row):
        if word_count == 0:
            res[j] = 0
            continue

        tf = col / word_count
        has_word_count = col_sums[j]

        idf = math.log(((doc_count + 1) / (has_word_count + 1))) + 1

        res[j] = tf * idf

    total_distance = euc_distance(res, len(res))

    for i in range(len(row)):
        if total_distance == 0:
            res[i] = 0
        else:
            res[i] /= total_distance

    return res
```

Figure 86: The tf-idf sub function

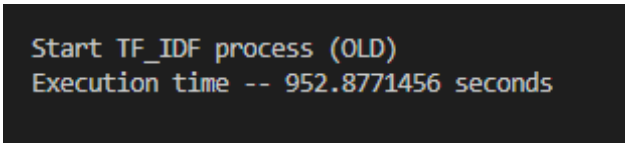


Figure 87: Execution time of the normal tf-idf algorithm

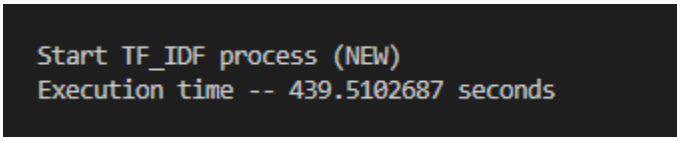


Figure 88: Execution time of the multi-threaded tf-idf algorithm

The execution time of the process has decreased to at least half of the original time here, which is twice slower than the ideal run time. Nonetheless, it is still an improvement compared to the original implementation.

Building the Model

In the project, the most computationally expensive section is building the Self - Organizing model. The SOM's complexity is noted as **O(S^2).** The building of a 4 x 4 lattice with 8000 iterations and a 0.05 learning rate ran for 10 minutes and 40 seconds. As the lattice size and iteration increase, the time of building would also increase due to the increase in the number of clusters needed.

Clustering the Tweets

The clustering of tweets is when the researchers assign a cluster to a specific tweet. Here, the process is the same as running one epoch, except that the BMU is then assigned to the tweet. In the project, it took 25 mins and 30 seconds for the model to cluster the tweets.

CHAPTER 4

SUMMARY, CONCLUSION AND RECOMMENDATION

Summary of Findings

By training a SOM model using our scraped tweets, the researchers were able to discover several topics that are spread over sixteen clusters. These topics are about (1) the implementation of face-to-face in general for clusters 1-6; (2) performance tasks of students for clusters 7-8; (3) government response to the new standard and online classes in clusters 9-12; (4) college experience of students in general for clusters 13-15; and a meaningless or uninterpretable topic for cluster 16.

.

Tweet ID	1	2	3	4	5	6	7	8	9	10
Cluster 1	ang	class	lang	limited	pilot	say	school	start	student	teacher
Cluster 2	class	deped	like	limited	need	pilot	school	start	student	teacher
Cluster 3	class	deped	help	like	limited	say	school	student	study	tomorrow
Cluster 4	area	confuse	government	help	look	love	plan	run	study	talk
Cluster 5	class	go	good	miss	pilot	school	start	student	teacher	today
Cluster 6	class	deped	limited	need	pilot	say	school	student	time	want
Cluster 7	believe	best	break	department	level	life	provide	tomorrow	video	watch
Cluster 8	area	dengan	grow	help	level	monday	parang	ready	study	tomorrow
Cluster 9	day	go	good	miss	run	school	start	student	today	work
Cluster 10	country	cutie	floor	get	have	high	know	mga	run	think
Cluster 11	allow	attend	break	follow	get	nga	problem	rodrigo	run	sleep
Cluster 12	agad	atleast	big	case	com	drain	internet	meeting	new	share
Cluster 13	ako	college	course	day	friend	hahaha	health	new	test	training
Cluster 14	ako	child	day	friend	get	have	morning	new	run	set
Cluster 15	blend	grabe	have	love	morning	second	soon	state	walang	wifi
Cluster 16	commission	company	long	lot	manage	mga	morning	okay	philippine	world

Figure 89: The clusters and its keywords

In the figure above, the clusters and their top keywords are shown. The top keywords are the hints for the researchers to discover what the cluster might be. While the researchers were able to derive such topics from the keywords, it is still important to note that it might still be improved by a true expert in the field, particularly in the education field, and eliminate any subjectivity in the interpretations. Also, much noise and general terms are present in the top keywords. The researchers compensated for this by finding the keywords that have the most meaning by their discernment. The table below will show the results the researchers gathered for every topic.

Topic	Clusters	Positive	Negative	Neutral	Total
Implementation of face-to-face	1-6	1677 **(46.3%)**	1262 **(34.6%)**	698 **(19.1%)**	3637
Performance tasks	7-8	1311 **(39.1%)**	1504 **(44.9%)**	534 **(15.9%)**	3349
Government efforts or response	9-12	3261 **(39.8%)**	2750 **(33.6%)**	2164 **(26.4%)**	8175
College experience in general	13-15	1560 **(54.4%)**	573 **(19.9%)**	734 **(25.6%)**	2,867
No topic	16	934 **(54.7%)**	373 **(21.8%)**	400 **(23.4%)**	1,707

Figure 90: Cluster Interpretation and Sentiment Count

Most of our scraped data is clustered around government efforts and their response to using online classes as a means of education. The tweets of the most populated cluster resulted in a positive response, while negative responses were not so far behind. The tweets on this cluster could be influenced by politics since the topic is about government response. This topic also has the most significant neutral percentage. This could be because some data could be coming from official broadcast media accounts, which generally tend to be neutral in sentiment.

An exciting discovery is about the topic of performance tasks. Among all the clusters, this topic has relatively the most significant negative percentage. This could mean that the public is unsatisfied with the performance tasks given through online classes. This could be something to consider by educational institutions and be later improved in the future. On the contrary, the college experience has the most significant positive percentage out of all the topics, excluding the meaningless one.

Conclusions

The dataset, lattice size, and iterations are the key factors in getting the best topics in the Self-Organizing Map. The SOM performs well when given a large dataset and is cleaned and pre-processed. The model creates the topics out of a given dataset accurately. However, the researchers found out that it is not suitable for clustering data on Twitter since it is a microblogging social network thus, it is prone to typographical errors. Some tweets may contain a mix of English and non-English words that could make the dataset even more inaccurate when cleaned. A dataset with formal words and fewer typographical errors would be optimal for the SOM model.

The pre-processing and cleaning part is very important for getting the best results in the SOM model to avoid the model from making senseless topics. Chunking the Tweets first before applying a sentiment analysis will be best for getting the right sentiment in a Tweet. There may be cases where the SOM model will only cluster to one side of the map or even to only one topic. This is a result of non-optimal hyperparameter tuning or a noisy dataset.

Recommendations

Tweets can be a medium for opinion-based data mining, and cover different genres based on the covered data. However, this data is limited since the free scraper can only cover a small number of Tweets in a week. This can vastly reduce the model's accuracy since it gives us a small dataset. A recommended alternative would be to get an approved Tweet dataset online for training purposes and check for validity.

Twitter is a microblogging social media platform; therefore Tweets are prone to typographical errors and sometimes meaningless data. The preprocessor must be tested many times for optimal results. The researchers recommend Reddit, which is a social news aggregation and also a platform for discussions, as a source for data sets since most of the data, although not all, is much more formal and cleaner.

Our system is only catering to English Tweets. The researchers would recommend translating non - English Tweets to English to make the system more flexible. This will also increase the data set that the entire population is catered to and cover the most opinion in a population. Since the SOM model as a way to do topic modeling is relatively new, the researchers would recommend to use our study as a basis further and find possible ways to use the algorithm the researchers are using.

BIBLIOGRAPHY

[1]	Alcober, G. M. I., & Revano, T. F. (2022, March 16). Twitter Sentiment Analysis towards Online Learning during COVID-19 in the Philippines. IEEE Xplore. https://ieeexplore.ieee.org/document/9731999
[2]	Mujahid, M., Lee, E., Rustam, F., Washington, P. B., Ullah, S., Reshi, A. A., & Ashraf, I. (2021, September 21). Sentiment Analysis and Topic Modeling on Tweets about Online Education during COVID-19. MDPI. Retrieved May 26, 2022, from https://www.mdpi.com/2076-3417/11/18/8438
[3]	Ponmalai, Ravi, & Kamath, Chandrika. Self-Organizing Maps and Their Applications to Data Analysis. United States. https://doi.org/10.2172/1566795
[4]	Kohonen, T. (1990, September). *The Self-Organizing Map*. IEEE Xplore. Retrieved May 21, 2022, from https://sci2s.ugr.es/keel/pdf/algorithm/articulo/1990-Kohonen-PIEEE.pdf
[5]	Allen, M., & Cervo, D. (2015). *Multi-Domain Master Data Management*. Elsevier Gezondheidszorg.
[6]	Johnston, D., Oh, J., Lynch, B., Oh, J., Lowing, S., Houcheime, W., & Klepfish, N. (2019, December 29). What Is Scraping | About Price & Web Scraping Tools | Imperva. Imperva. Retrieved May 21, 2022, from https://www.imperva.com/learn/application-security/web-scraping-attack/#:%7E:text=Web%20scraping%20is%20the%20process,replicate%20entire%20website%20content%20elsewhere.
[7]	Guide To Data Cleaning: Definition, Benefits, Components, And How To Clean Your Data. (n.d.). Tableau. Retrieved May 21, 2022, from https://www.tableau.com/learn/articles/what-is-data-cleaning
[8]	Jha, A. (2021, December 31). Vectorization Techniques in NLP [Guide]. Neptune.Ai. Retrieved May 21, 2022, from https://neptune.ai/blog/vectorization-techniques-in-nlp-guide
[9]	Signh, A. K., & Shashi, M. (2019). Vectorization of Text Documents for Identifying Unifiable News Articles. Department of Computer Science & Systems Engineering, Andhra University, India. Retrieved May 21, 2022, from https://pdfs.semanticscholar.org/caf5/b10072c03fc78b4d4a5c007c8e9e1feaa0d4.pdf

[10]	Understanding TF-IDF for Machine Learning. (2021, October 7). Capital One. Retrieved May 21, 2022, from https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/
[11]	I. Douven and W. Meijs.(2007). Measuring coherence. Synthese, 156:405–425.
[12]	Kapadia, S. (2021, December 11). Evaluate Topic Models: Latent Dirichlet Allocation (LDA). Medium. Retrieved May 22, 2022, from https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0
[13]	Mimno, D., Talley, E., Wallach, H., Leenders, M., & McCallum, A. (n.d.). Optimizing Semantic Coherence in Topic Models. Association for Computing Machinery. Retrieved May 22, 2022, from http://dirichlet.net/pdf/mimno11optimizing.pdf
[14]	Sentiment Analysis – The Lexicon Based Approach. (2021, March 2). Top Microsoft Dynamics and NetSuite Partner & Dynamics CRM Consultant in San Diego. Retrieved May 20, 2022, from https://www.alphabold.com/sentiment-analysis-the-lexicon-based-approach/#:%7E:text=Rule%20based%20sentiment%20analysis%20refers,with%20their%20corresponding%20intensity%20measure
[15]	sklearn.metrics.silhouette_score. (n.d.). Scikit-Learn. Retrieved June 4, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html
[16]	Kapoor, Akshay & Jindal, Veni. (2020). Exploring Self Organizing Maps for Brand oriented Twitter Sentiment Analysis. 10.13140/RG.2.2.22212.45440.
[17]	Relia, K., Akbari, M., Duncan, D., & Chunara, R. (2018, November). Socio-spatial Self-organizing Maps: Using Social Media to Assess Relevant Geographies for Exposure to Social Processes. Association for Computing Machinery. Retrieved May 20, 2022, from https://dl.acm.org/doi/pdf/10.1145/3274414
[18]	Thorsten Joachims.(1998). Text categorization with support vector machines: Learning with many relevant features. In European conference on machine learning. Springer, 137–142.
[19]	Arambepola, N. (2020). Analysing the Tweets about Distance Learning during COVID-19 Pandemic using Sentiment Analysis. Faculty of Computing and Technology - University of Kelaniya, Sri Lanka. Retrieved June 1, 2022, from https://fct.kln.ac.lk/media/pdf/proceedings/ICACT-2020/F-7.pdf

CURRICULUM VITAE

PERSONAL INFORMATION

Name	Rhyan Augustine D. La Rosa
Date of Birth	November 2, 1999
Address	Modena Subdivision, Tunghaan, Minglanilla, Cebu
Status	Single
Citizenship	Filipino
Religion	Roman Catholic
Email Address	rhyan.larosa@gmail.com
Contact Number	09177703050

ACADEMIC INFORMATION

Tertiary		University of San Jose-Recoletos
Secondary	**Senior High**	University of San Jose-Recoletos
	Junior High	Don Bosco Technical College
Primary		Don Bosco Technical College

PROFESSIONAL SKILLS

Language	JavaScript, C, Dart, Java, Python
Software	Git, Visual Studio, Apache NetBeans
Operating Systems	Windows OS, MacOS

CURRICULUM VITAE

PERSONAL INFORMATION

Name	Redempto D. Legaspi III
Date of Birth	October 9, 1999
Address	529-B Cebu South Rd., Brgy. Kinasang-an, Cebu
Status	Single
Citizenship	Filipino
Religion	Roman Catholic
Email Address	redemptolegaspi@gmail.com
Contact Number	0908 813 7824

ACADEMIC INFORMATION

Tertiary		University of San Jose-Recoletos
Secondary	**Senior High**	Cebu Institute of Technology-University
	Junior High	Don Bosco Technical College
Primary		St. Paul Learning Center Incorporated

PROFESSIONAL SKILLS

Language	Javascript, Dart, Ruby, C, Python, PHP
Software	Git, Flutter, React, Laravel, Ruby on Rails
Operating Systems	Windows OS

CURRICULUM VITAE

PERSONAL INFORMATION

Name	John Paul B. Lopez
Date of Birth	October 5, 1999
Address	049 Purok 19, Colon, Naga, Cebu
Status	Single
Citizenship	Filipino
Religion	Roman Catholic
Email Address	jpblopez05@gmail.com
Contact Number	0926 973 3058

ACADEMIC INFORMATION

Tertiary		University of San Jose-Recoletos
Secondary	**Senior High**	University of San Jose-Recoletos
	Junior High	Sacred Heart Diocesan School Inc.
Primary		Mahayag SPED Center

PROFESSIONAL SKILLS

Language	Javascript, PHP, Python
Software	Git, AWS, MongoDB, MySQL, RabbitMQ
Operating Systems	Windows OS, Unix-based OS e.g Linux, OSX

UNIVERSITY OF SAN JOSE-RECOLETOS
Center for Policy, Research and Development Studies

CERTIFICATION

The manuscript entitled **"MINERVA: OPINION MINING OF TWEETS ABOUT E - LEARNING USING SELF - ORGANIZING MAPS"** has undergone Similarity and Grammarly tests under Turnitin and Grammarly softwares.

AUTHOR/s: **LA ROSA, RHYAN AUGUSTINE D.**
LEGASPI III, REDEMPTO D.
LOPEZ, JOHN PAUL B.

TURNITIN RESULT: **7%**

GRAMMARLY RESULT: **97/100**

This is to certify further that the manuscript has registered an originality grade of **93%** and technical writing quality of **97%** which includes grammar, spelling, and punctuations, among others. Given this 17th day of June, 2022 at the Quality Assurance Unit of the Center for Policy, Research and Development Studies, University of San Jose-Recoletos, Cebu City.

AGNES C. SEQUIÑO, Ph.D.
CPRDS Director