



African Leadership University

Computer Science Capstone

Capstone Title: A Deep Learning Tool to Bridge the Situational Awareness Gap in Disaster Management

Author's Name: Kelvin Kinda Chumbe

Supervisor's Name: Tatenda Duncan


Year of Graduation: 2021

Declaration

I certify that the work presented in this capstone is my own and that any work that has been performed by others is appropriately cited.

Date: 4th May 2021

Name: Kelvin Chumbe

Signature: 

Acknowledgement

I would first like to thank my advisor, Mr. Tatenda Kavvu, whose expertise and experience were invaluable in the shaping of the project. Your support and feedback pushed me to sharpen my thinking and expound my perspective of the research topic.

I'm would also like to appreciate the Computer Science faculty at the African Leadership University for their continued support and mentorship.

I would like to appreciate my colleagues and friends who pushed me to deliver the best work I could. I'm very grateful for their support both technically and emotionally during this project.

Finally, I would like to appreciate my family for their love, prayers and support, both financially and emotionally. I'm very grateful for your counsel and guidance.

Abstract

Early stages of a disaster are very critical in understanding the nature of events that are happening. It is paramount that disaster management teams gain a situational awareness of prevailing events so they can better position themselves when responding to the disaster. During this early stage, disaster management teams might be faced with one of two scenarios: lack of information about events that are unfolding or too much information about the disaster streaming in. Both scenarios present a great challenge to disaster management teams as they impede their ability to quickly gain situational awareness of the disaster and respond accordingly. In this project, we explore social media, particularly Twitter, as a source of disaster-related information. We build a tool consisting of a Machine Learning pipeline to extract key information from tweets and a web-based application to deploy the trained models. The Machine Learning pipeline consists of a data collection and preprocessing component and two Machine Learning models: a filtering model and a categorizing model, that extract the information from the tweets. The filtering model identifies whether a tweet provides information related to a disaster or if the information is irrelevant. The categorizing model then classifies relevant tweets into one of 5 humanitarian response activities categories i.e. Casualty, Damage Report, Relief Request, Caution and Advice, and Sympathy and Support. We built a web application to summarize and report the insights from the Machine Learning pipeline. We extract location information from the tweets and plot each tweet on a crowdmap to serve as a Common Operational Picture (COP).

KEYWORDS:

Social Media, Situational Awareness, Disaster, Disaster Management, Machine Learning Pipeline, Crowdmap

Table of Contents

Declaration	2
Acknowledgement	3
Abstract	4
Table of Contents	5
Table of Figures	7
List of Tables	9
List of Acronyms and Abbreviations	10
Chapter 1: Project Introduction.....	11
1.1 Introduction and Motivation.....	11
1.2 Problem Statement	12
1.3 Project Aim	12
1.4 Project Objectives	12
1.5 Project Scope.....	13
1.6 Technical Requirements.....	13
1.7 Project Limitations	14
Chapter 2: Literature review	16
2.1 Introduction	16
2.2 Context Literature	16
1. Crowdsourcing	16
2. Social Media as a tool for information propagation.....	17
3. Tweet Classification	17
Conclusion.....	18
Chapter 3: Requirement Gathering and System Analysis.....	20
3.1 Introduction	20
3.2 Methodology	20
Data	21
3.3 Data Collection	22
3.4 Software Development Process.....	23
3.5 System Description	25

Functional Requirements:	25
Non-Functional Requirements	25
Chapter 4: System Design	26
4.1 Introduction	26
4.2 Architecture	26
4.3 Platform	30
4.4 Programming	31
1. Data Analysis and Modelling	31
2. Front-End Development	32
3. Back-End Development	32
4.5 Security	33
4.6 Model and Graphical User Interface Design	33
4.6.1 Filtering Model	33
4.6.3 Web App Design	46
Chapter 5: Implementation and Testing	55
5.1 Introduction	55
1. Filtering Model Results	56
2. Categorizing Model Results	60
5.3 System Accuracy	64
5.4 Functional Testing	66
1. Unit Testing	66
5.5 Analysis and Discussion	71
5.6 Recommendations	71
5.7 Future Work	72
5.8 Conclusion	72
References	74

Table of Figures

Fig 1. Agile Software Development Life Cycle	24
Fig 2. UML Use Case Diagram	27
Fig 3. Level 2 Data Flow Diagram	28
Fig 4. UML Sequence Diagram.....	29
Fig 5. Data collection and preprocessing pipeline for the filtering model	34
Fig 6. Some of the word contractions and their expanded equivalents	35
Fig 7. Preprocessing steps applied to the tweets.....	35
Fig 8. Steps to tokenize, generate a word_index vocabulary and generate sequences	36
Fig 9. Creating a dictionary of GloVe embedding vectors	37
Fig 10. Building a weight matrix from the word_index vocabulary and GloVe embedding vocabulary	37
Fig 11. CNN model architecture for the filtering model	38
Fig 12. BiLSTM model architecture for the filtering model	38
Fig 13. BERT_LSTM_ATTENTION model architecture for the filtering model	39
Fig 14. BERT_LSTM_ATTENTION_Tuning model architecture for the filtering model.....	40
Fig 15. Data collection and preprocessing pipeline for the categorizing model	41
Fig 16. Sample tweets with their Google Translate translation.....	42
Fig 17. Base model for the categorizing model. Uses 100-dimensional embedding vectors	43
Fig 18. BiLSTM (200d) model architecture for the categorizing model. Uses 200-dimensional embedding	44
Fig 19. BERT_LSTM_2_Attention model architecture for the categorizing model.....	44
Fig 20. BERT_LSTM_4_Attention model architecture for the categorizing model.....	45
Fig 21. BERT_LSTM_Attention_Tuning model architecture for the categorizing model	45
Fig 22. The homepage of the application features a form to collect information to stream the tweets.	47
Fig 23. A loading icon is displayed as the app fetches data from the API	47
Fig 24. The dashboard page showing the crowdmap and a section of the tweets list	48
Fig 25. Zoomed in image of the crowdmap.....	49
Fig 26. Zoomed in image of a tweet location showing the surrounding area, landmarks and roads	49
Fig 27. Tweets list section. Tweets are displayed with the author, date, tweet text and predicted category.....	49
Fig 28. Image showing the category distribution of tweets.....	50
Fig 29. Processing of request object body information and initialization of our custom thread object.....	52
Fig 30. Creation of the GeoJSON object using the tweet's information	54
Fig 31. A snippet of the bounding boxes for all countries.....	54
Fig 32. Plot of model accuracy, precision, recall and f1-scores of the different model architectures	57

Fig 33. Confusion matrix for the CNN model.....	58
Fig 34. Confusion matrix for the BiLSTM model.....	58
Fig 35. Confusion matrix for the BERT_LSTM_ATTENTION model.....	59
Fig 36. Confusion matrix for the BERT_LSTM_ATTENTION_Tuning model	59
Fig 37. Plot of model accuracy, precision, recall and f1-scores of the different model architectures	61
Fig 38. Confusion matrix for the Base Model (BiLSTM).....	62
Fig 39. Confusion matrix for the BiLSTM (200d) model	62
Fig 40. Confusion matrix for the BERT_LSTM_2_Attention model	63
Fig 41. Confusion matrix for the BERT_LSTM_4_Attention model	63
Fig 42. Confusion matrix for the BERT_LSTM_Attention_Tuning model.....	64
Fig 43. Sample input included in the body of the request to the API.....	67
Fig 44. Sample response from the streaming module.....	68
Fig 45. User input page showing user input fields and sample input	69
Fig 46. Dashboard showing data filters as an option list to restrict input	69
Fig 47. Response received by the display module from the streaming module	70
Fig 48. Display module visualizing the received data from the streaming module	70

List of Tables

Table 1. Estimated Project Budget.....	15
Table 2. Accuracy, precision, recall and f1-scores of the different model architectures of the filtering model	57
Table 3. Accuracy, precision, recall and f1-scores of the different model architectures of the categorizing model.....	60

List of Acronyms and Abbreviations

Abbreviation	Meaning
GloVe	Global Vectors for Word Representation
CNN	Convolution Neural Network
LSTM	Long Short-Term Memory
BiLSTM	Bidirectional Long Short-Term Memory
DFD	Data Flow Diagram
NLP	Natural Language Processing
COP	Common Operational Picture
BERT	Bidirectional Encoder Representations from Transformers

Chapter 1: Project Introduction

1.1 Introduction and Motivation

In recent years, we use technology to identify early warning signs and predict the occurrence of disasters, simulate their progression and calculate an estimation of the damages to be incurred were the event to happen. Understanding how, when and where these disasters occur is crucial in managing their impact. This makes disaster management a very integral part of our lives. During a disaster, response and recovery operations need to be carried out with speed and should be done efficiently. To increase efficiency throughout the team, everyone must have a similar understanding of the current situation, its dynamic nature and any changes to the current prevailing situation. This is the team's shared situational awareness [1].

Early stages of a disaster are characterized by a dynamic and rapidly evolving environment. This makes it extremely difficult to assess the initial conditions of a disaster, especially when the information available to make decisions is limited. Understanding these initial conditions is an important step in figuring out the possible causes of the current events and determining the direction in which the events are evolving. Establishing situational awareness as early as possible during a disaster is a very crucial step in planning for the mitigation, response and recovery of the disaster [2]. Lack of adequate situational awareness in a disaster makes search and rescue operations and damage assessments quite difficult for any teams sent on-site due to the lack of up-to-date information on the disaster [3]. To get wind of the status quo and appropriately distribute resources to affected areas, disaster management teams need to work with sufficient information on the disaster, affected areas, damages and losses, affected people, areas to be evacuated and available resources [4].

Social media and microblogging platforms, particularly Twitter, are a suitable alternative source to provide information on victims, their needs and their situation in real-time. Social media streams are considered important due to the fast and frequent localised updates on disaster events [5]. Due to the very nature of social media, people can express their views in real-time to a multitude of people. This real-time nature makes it an important tool for disaster management teams as they can relay crucial information to the public, acquire their status information, gather sensitive

information about a disaster from the public, facilitate early warnings detection and help coordinate relief efforts [6], [7].

1.2 Problem Statement

Slow flow of information to humanitarian organisations during early stages of a disaster can stall their response to the disaster. With the dynamic nature of events at this stage of the disaster, this could have negative effects on the entire disaster response operations. Despite this, affected individuals share a lot of information about their status or their environment on social media platforms. This information is mostly shared with the intention of updating their loved ones about their current status or informing others about current events. This makes social media a potential alternative source of disaster-related information that humanitarian organizations can turn to for real-time and localised updates to gain situational awareness of prevailing events. Since this information is already in the public domain, humanitarian organisations can use it to influence their response strategies towards the disaster.

1.3 Project Aim

The main aim of this project is to build a Machine Learning tool that retrieves tweets from Twitter in real-time, identifies the relevance of the message in the tweet and tags it as one of 6 humanitarian activities categories. The tool then identifies the location of the tweets and gives a basic report of prevailing events as retrieved from the tweets.

1.4 Project Objectives

1. Collect data from multiple data sources i.e. Twitter API, CrisisLex and CrisisNLP and preprocess it.

2. Build a Machine Learning pipeline with two models: the first filters out tweets based on their informativeness with regards to a disaster and the second classifies the tweets into one of 5 categories depending on the message and the theme of the tweets
3. Build a dashboard/ crowdmap that reports on insights gathered through the machine learning pipeline

1.5 Project Scope

This project will only focus on Twitter as a social media platform on this project. This is because Twitter provides a publicly accessible API to access real data on the platform. Only tweets written in English will be used in the analysis when training and testing the models. The project will only focus on one stage of disaster management i.e. disaster response although findings can be used to inform other stages of disaster management including disaster preparedness and disaster recovery. Only tweets with their geotag information included and tweets with location information embedded in the message of the tweets will be used in the location extraction stage to train the Named Entity Recognition tagger. Location estimation of tweets without any location information directly provided will not be conducted.

1.6 Technical Requirements

1. Data collection and preprocessing
 - Twitter API
 - Tweepy
 - Python
2. Data preprocessing, feature engineering and analysis
 - Pandas
 - Numpy

- Matplotlib
- Seaborn
- Scikit Learn
- NLTK

3. Data Modelling

- Tensorflow

4. Deployment

- Heroku
- React JS
- Flask

1.7 Project Limitations

The main potential limitation of this project is the lack of adequate labelled data. Since we are using predefined categories to carry out our classification tasks, there is a likelihood that currently available datasets on the disasters that we'll be exploring may not have the data labelled using these categories. A possible workaround this challenge is to manually annotate these datasets with the categories that we've selected. This would be a time-intensive and costly process since we would need to hire volunteers to do the job.

1.8 Project Time Frame

[Link to Gantt Chart](#)

1.9 Project Budget

Table 1. Estimated Project Budget

Item		Quantity	Unit Price	Total (\$)
Premium	Twitter			
Developer Account		1	\$289	\$289
Volunteer	data			
annotators		2	\$100	\$200
			Total	\$489

Chapter 2: Literature review

2.1 Introduction

During disasters, people turn to social media platforms to share and retrieve information about the ongoing disaster [6]. This makes social media a very important communication tool during disasters to disseminate information to the masses and send victims' information to disaster response teams to aid in the coordination of relief efforts [6]. Previous research into the use of social media platforms, Twitter in particular, in disaster management is manifold. Researchers have studied the use of Twitter as a crowdsourcing tool to gather disaster-related information (both images and textual messages) during a disaster [5], [3]. Research has also been done to build models to correctly categorize and classify Twitter posts from disaster regions into user-defined categories or schemas to aid in disaster response operations [8], [9], [10]. Other researchers have created tools and platforms to capture situational awareness reports based on social media activity during a disaster [11], [12]. Researchers have also studied how microblog posts can be extracted to support common situational awareness among disaster response teams [13]. Other works have been on understanding how social media platforms work and what features are essential to information diffusion during an emergency, particularly for Twitter. Such features include retweets, hashtags, following, tagging and geotagging [14].

2.2 Context Literature

1. Crowdsourcing

Social Media applications offer a powerful capability for data collection and visualization during disasters for decision making. Crowdsourcing allows multitudes of people to play a role in disaster response management. This could be as simple as posting about an event or validating a piece of information or media, normally seen through the number of reactions on a post [15]. Crowdsourcing information works on the principle that information that has many people interested and posting about is closer to the ground truth than information with less interest [16]. This is a form of collective wisdom (wisdom of the crowd) that leverages social participation to provide and validate the information. Social Media provides an open and convenient platform for

collecting real-time information during a disaster [15]. One advantage of social media crowdsourcing is the ability to include geotag information along with the post. This data can help humanitarian organizations optimize their search and rescue operations since they can accurately locate help requests by victims [15]. Geotag information is, however, not a reliable source of location information. This is because of the circumstance related to a disaster. Victims might decide to turn off their location feature on their phone to reduce their phone power consumption and the resulting post might not have geotagged information. Allowing victims to add their information along with the body of the post offers a much reliable source for location data.

2. Social Media as a tool for information propagation

According to [17], the retweet feature on Twitter is very important when investigating information propagation behaviours by people before, during and after a crisis. The authors state that the retweet feature was commonly used as an information recommendation system to propagate information that one finds valuable and deems important for other people in their network to know. In their work, they reported that for the majority of retweeted posts, their original authors were likely to be people who were geographically local to the event. This sheds some light on the role of people who are local to the event in the dissemination of information both to the general public, humanitarian organizations and the government. They also observed that most of the retweeted posts among the locals in the Oklahoma Fires contained very specific emergency-related information that they deemed important and locally relevant for other locals to know. This includes information on places to shelter and people's observations of the emergency and how it progressed. From their observations, we can ascertain that the role of those local to the event is very crucial in the flow of real-time information about a disaster since they can either be the source of it or be the propagators of emergency-related information to other locals and to the general public [14].

3. Tweet Classification

Using information from social media during a disaster can provide accurate insights on the events happening and their impact if used correctly. These insights can provide response teams and

humanitarian organizations with accurate information to work with in advance of the official communications from relevant bodies. Classification of tweets during a disaster can help in determining the current situation as is experienced in the affected regions and detect the current events as they are happening. This information can help disaster response teams mobilize the appropriate personnel to react timely and effectively when it comes to search and rescue operations and can help the teams organize other crowdsourcing initiatives to supplement their resources [18]. This is evident even during the Haiti earthquakes, where the Red Cross organization was able to raise \$8 million in just 48 hours due to the powerful propagation of information on social media platforms about the crisis that was ongoing and the crowdsourcing initiatives that had been launched [15]. Despite the presence of useful information in social media posts, they are often not well organized to identify the information in them. It would be very difficult to use human resources to extract this information from a multitude of tweets during a disaster hence the need to employ technology to quickly classify (e.g. rescue needed, water needed, casualty, sick, floods, power/ electricity in the case of a flooding disaster) and assign relevant personnel and resources [18].

Conclusion

From the existing body of literature on the use of social media during disasters, we observe social media as a very important tool in the propagation of information both from victims to their families and from humanitarian agencies and governments to the general public during and after the onset of a disaster. We also observe social media as an alternative source of disaster-related information for humanitarian organizations and disaster response teams seeking to gain situational awareness on prevailing events. This can be done through crowdsourcing data from the platforms. Machine Learning models have been developed to help in the classification of disaster-related tweets to determine if a tweet's message is related to a disaster or is just noise. Models have also been built to classify tweets into humanitarian response categories. A gap, however, exists in the development and deployment of these models. Most models that have been built are only for research purposes and haven't been integrated with a tool to ensure the usability of the system or deployed. My proposed solution is to build and deploy a tool that encompasses a machine learning

pipeline to preprocess tweets and classify them into one of 5 humanitarian response categories, analyze the tweets and give an interactive report on the status quo on the ground as disaster teams plan their operations.

Chapter 3: Requirement Gathering and System Analysis

3.1 Introduction

We propose a tool, a web application that incorporates a machine learning pipeline that retrieves information from Twitter, processes it and feeds it to a natural language processing (NLP) pipeline for text classification and named entity recognition. The pipeline is composed of four main modules:

1. Data collection
2. Data preprocessing
3. Filtering model
4. Categorizing model

The first stage in the pipeline is the retrieval of tweets. The final pipeline features real-time tweet retrieval using the Twitter API and Tweepy library. This allows us for quick and efficient authentication and authorization to access the API and pull out the required resources. Tweets will be taken through a series of preprocessing steps to reduce the noise in the data and make it ready to be fed into the NLP pipeline. After preprocessing, named entities present in the tweets will be extracted. Particularly, we'll be interested in identifying any location information from the tweets. We'll then pass the preprocessed tweets to a filtering model. This model identifies any informative tweets, i.e. tweets that give us information about a disaster and filters out non-informative tweets. Filtered informative tweets will then be fed to a categorizing model which classifies the tweets based on 6 humanitarian activities categories (Casualty, Search and Rescue, Damage Report, Relief Request and Donations, Caution and Advice, Sympathy and Support, Other Information). Insights obtained from the pipeline will then be reported as a dashboard on the web application.

3.2 Methodology

This project will be using both qualitative and quantitative analysis.

Datasets

To develop the Machine Learning models, and evaluate their performance, supervised machine learning approaches are employed. Thus, data to train the machine learning models needs to be annotated with the correct labels. Annotated tweets are collected from [CrisisLex](#) and [CrisisNLP](#) repositories which contain large collections of data collected through the Twitter API but have been labelled for supervised machine learning. This project makes use of several datasets collected from the repositories:

1 CrisisLexT26 [19]

This is a collection of tweets collected from 26 disasters that occurred between 2012 and 2013. The tweets are labelled based on their informativeness, source and information type.

2. CrisisNLP [20]

This is a collection of tweets collected from different disasters that occurred between 2013 and 2015 labelled using different categories depending on the nature of the disaster

3. SWDM2013_dataset [21]

4. ISCRAM2013_dataset [22]

Machine Learning Models

1. Filtering Model

This model is a Bidirectional Long Short-Term Memory (BiLSTM) model architecture. LSTMs are designed to identify long-range dependencies within a text, i.e. words that are way in the past can influence the prediction of the current word. With BiLSTMs, words that are way in the future can influence the prediction of the current word.

The model will consist of four BiLSTM layers, each with 32 units. The BiLSTM layers will then be connected to a Fully Connected Dense network with 2 layers to make the classification. The

first Dense layer will have 128 units, and the last layer will have 2 units (influenced by the number of categories) with a softmax activation.

2. Categorizing Model

The categorizing model also consists of multiple BiLSTM layers connected to Fully Connected Dense layers. For the classifier, we'll use four BiLSTM layers with 32 units each. For the Fully Connected Layers, we'll have 2 layers; one with 512 units and the other 6 units and a softmax activation.

Model Evaluation

The data collected will be divided by a 70-10-20 split into the train, validation and test sets. The train set will be the sample of the data used to train the machine learning models. The validation set will be the sample of the data we hold out during training to evaluate it. We'll tune our model's hyperparameters based on their performance on this holdout sample. The test set is also a holdout sample of the data that is not used in training. A final evaluation of the model will be done using this set. The model's performance on this set will be crucial as it will determine the overall model's performance for the tasks.

To evaluate the model, we'll use classification metrics since we are carrying out classification tasks. The key metrics that we will be focusing on are accuracy, precision, recall and f1 score. Each model in the pipeline (filtering and categorizing models) will be evaluated independently. This allows us to individually tune and optimize the subparts of the pipeline instead of tuning the entire pipeline at a go.

3.3 Data Collection

For data collection, we will be using the following tools to gather data. Below is a list of the tools and a description of each tool.

1. Twitter API

To retrieve tweets from Twitter once the tool is active, we will stream tweets using the Twitter API. This will allow us to plug into Twitter and download tweets from the platform in real-time.

2. Tweepy library

Tweepy provides an interface to quickly and easily authenticate, get authorization and retrieve tweets from the Twitter API. Using credentials received from the Twitter API developers account, we will authenticate ourselves on the platform using Tweepy and make API calls to retrieve data. Tweepy allows us to make specific requests to the API, including filtering for only tweets in English, tweets posted between a certain time range or tweets with a particular locality.

3. CrisisLex and CrisisNLP repositories

Since we'll be using a supervised learning approach to train our filtering and categorizing models, we'll collect labelled tweets from the CrisisNLP and CrisisLex repository. These repositories provide a large collection of human-annotated tweets from different disasters that have been used by other researchers working on similar challenges. Data from this repository is publicly available on the condition that credit is given to the authors/creators of the dataset.

3.4 Software Development Process

This project will employ the agile software development model. Agile development allows for focus on the main objectives and milestones of the project yet provides some room for improvement and changes along the way. At the start of the development phase, milestones and objectives are set out to build a series of modules that can be integrated into the main system by the end of the project.

The agile process flows as:

- Concept

The project ideation phase. This involved deciding which project to work on and the timelines involved for such a project

- Inception

This occurs in the early stages of the development phase. It involves ensuring some of the important requirements are set and fulfilled, e.g. system design and architecture, UI/UX designs, mockups etc.

- Construction/Iteration

After feedback review from the design phase, work is done to deliver a working product/ prototype based on the iteration's requirements and feedback

- Release

This phase involves the testing of the system, user documentation, address system defects and final release of the iteration into production

- Production

This involves continuous support and maintenance of the software after it is life

- Retirement

In this phase, the system is removed from production.

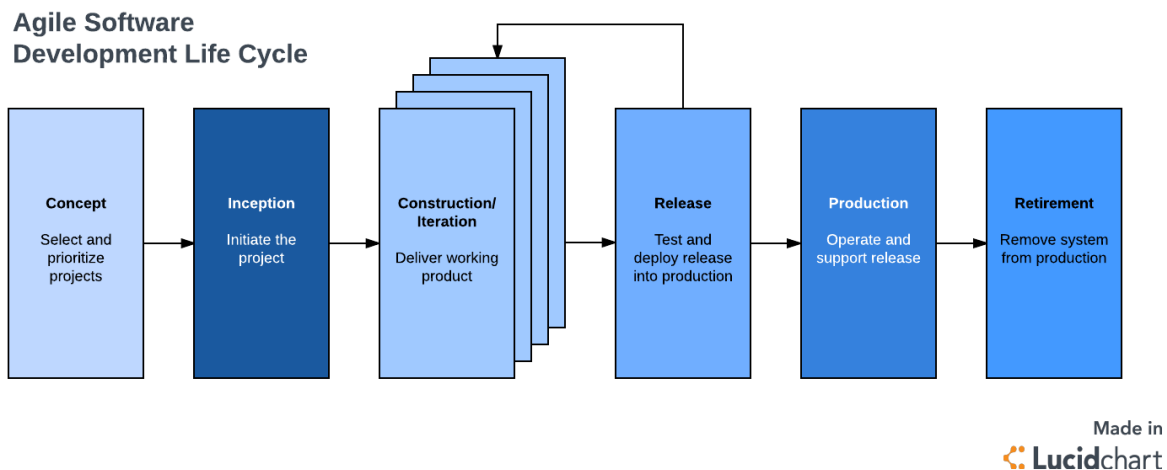


Fig 1. Agile Software Development Life Cycle [19]

3.5 System Description

Functional Requirements:

1. The streaming module must stream in tweets in real-time from using the Twitter API
2. Tweets will be streamed from the Twitter API in batches and temporarily stored for analysis.
3. Tweets will be preprocessed after they are received from the streaming module
4. The system must filter tweets based on relevance
5. The system must categorize tweets based on their messages
6. The system must analyze tweets to get insights and rules
7. The system must report insights to users in a clear and easy to understand dashboard
8. The dashboard should feature an interactive interface
9. The dashboard will be running of a web application
10. The web app will be deployed on Heroku

Non-Functional Requirements

1. Each batch of tweets streamed in will be 100 tweets in size.
2. Temporary tweets storage should not exceed 200MB
3. Dashboard will feature a minimalistic design
4. System must feature exception handling
5. System will display messages to provide visual feedback to the user

Chapter 4: System Design

4.1 Introduction

In the previous section, system requirements, both functional and non-functional, were identified. This created a model of how our tool would look like and what objectives it is to achieve. Users and actors in the system were identified, their roles defined and their requirements in the system. This was done by identifying different use cases in the system. The design of the system was represented using three architectural designs to aid in the development process:

1. UML use case diagram was used to map out the users' interactions with the system.
2. Data flow diagram was used to visualize the flow of data through the system
3. Sequence Diagram was used to show the user's interactions with objects in the system arranged in a time sequence

4.2 Architecture

The design of the system is described using 4 main diagrams: a UML use case diagram, Data flow Diagram, a Sequence Diagram and a system layout diagram. These were used to visualize the architecture of the system, visualize the flow of data and summarize users' interaction with the different objects of the system.

UML Use Case Diagram

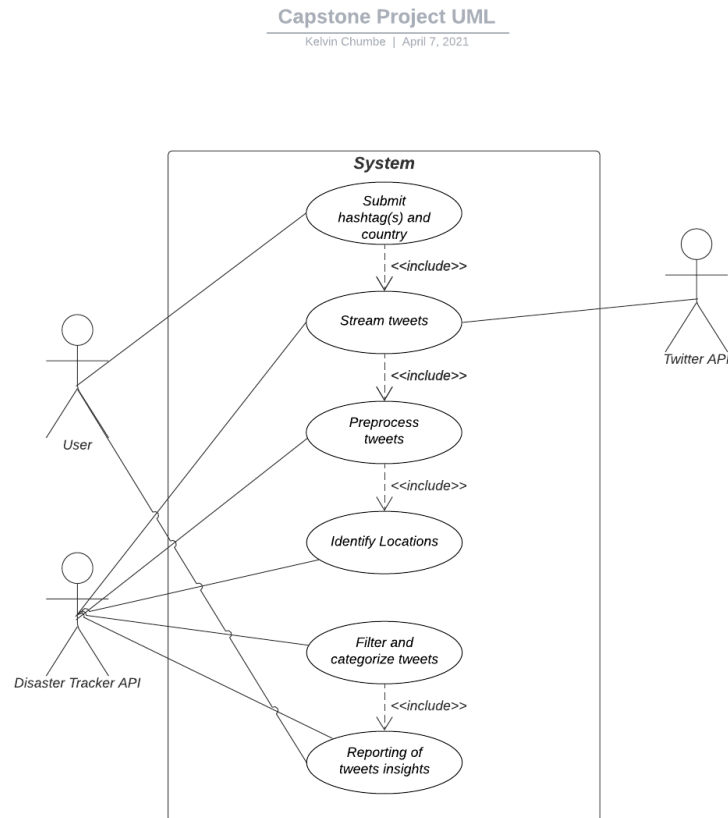


Fig 2. UML Use Case Diagram

The UML use case summarizes the interactions between the user and the system. In the UML use case above, the system is represented with 3 main entities i.e. the user, the Disaster Tracker API, our custom-built REST API, and the Twitter API. The user interacts with the system by submitting a hashtag. The system streams tweets using the hashtag as a filter. Tweets are then preprocessed, filtered, categorized and analysed then displayed to the user as a report.

Data Flow Diagram

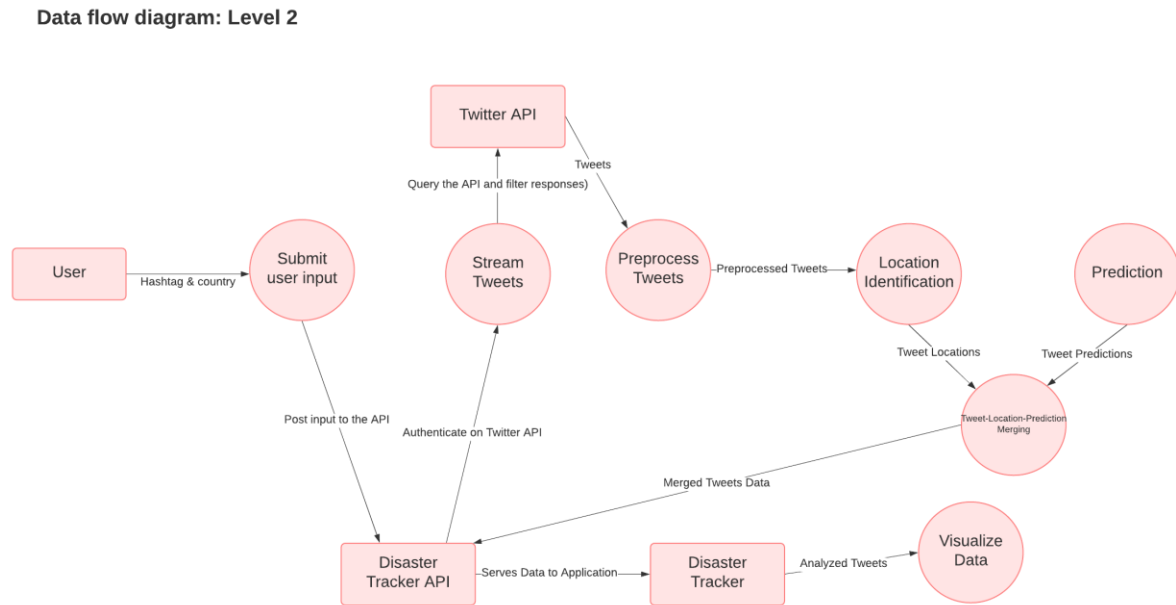


Fig 3. Level 2 Data Flow Diagram

The data flow diagram (DFD) summarizes the flow of data between the different entities and the different processes in the system. The diagram represents the data flow between 4 entities: the user, Disaster Tracker API, Twitter API and the Disaster Tracker application. The Disaster Tracker API is our custom-built REST API while the Disaster Tracker is the web-based dashboard. A user inputs a hashtag and a country which get submitted to the Disaster Tracker API. The API authenticates us on the Twitter API and queries the API and streams tweets applying a filter based on the hashtags and country submitted by the user. Tweets are preprocessed and their location information identified. Their categories are predicted and merged with the other tweets information. The Disaster Tracker API serves the tweets to the Disaster Tracker web app which analyses and visualizes the data to the user.

Sequence Diagram

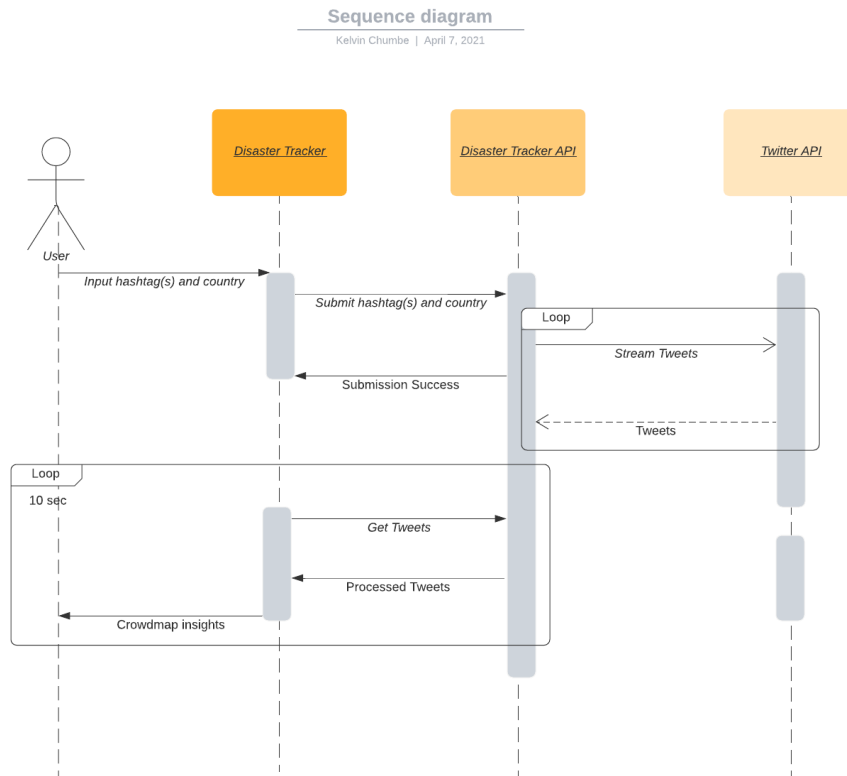


Fig 4. UML Sequence Diagram

The sequence diagram illustrates how actions are performed in the context of time. It describes the sequential execution of events and processes in the system. The first process is initiated by the user which is entering the hashtags and country they wish to stream tweets to the Disaster Tracker web app. The app submits the information to the Disaster Tracker API which sends a success message as a response to the request. The API streams tweets from the Twitter API. This is an asynchronous process hence does not wait for the results to be retrieved before sending a response. The process continues in a loop and does not automatically end its execution. The web app gets tweets from

the API and receives preprocessed tweets as a response. The step executes every 10 seconds in a loop.

4.3 Platform

The system is a web-based dashboard hence will be deployed to the web. Users will need to have access to a browser either using a PC, mobile device or any other device that has a browser to access it. Interactions with the system will however be limited to PCs and mobile devices. For a user to interact with the tool, they need to use a browser that supports JavaScript and has it enabled otherwise, they will only have view access to the tool and would be unable to interact with objects in the system.

Software Requirements:

- Operating System: Windows, Linux, macOS
- Programming Languages: Python, JavaScript, HTML, CSS
- Programming Frameworks/ Libraries: Flask, React.js, D3.js, Bootstrap
- Browser: Any
- Networking: Internet access

Hardware Requirements:

- Processor: Minimum of core i3
- Hard disk: Minimum of 500MB
- RAM: Minimum of 2GB
- Screen size: Any

4.4 Programming

1. Data Analysis and Modelling



Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for rapid application development, as well as for use as a scripting or glue language to connect existing components together.

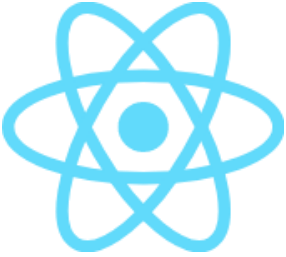


TensorFlow is an open-source software library for machine learning across a range of tasks. It is a system for building and training neural networks to detect and decipher patterns and correlations, analogous to (but not the same as) human learning and reasoning. It is used for both research and production at Google.



Scikit-Learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

2. Front-End Development



React is an open-source, front end, JavaScript library for building user interfaces or UI components.



D3.js is a JavaScript library for manipulating documents based on data. **D3** helps you bring data to life using HTML, SVG, and CSS. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

3. Back-End Development



Flask is a micro web framework written in python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where preexisting third-party libraries provide common functions.

4.5 Security

To ensure the web app is secure from external threats, some security measures are designed as the system is developed. User input is always validated against a defined set of rules. On the front-end, user input is validated before the web app submits the data to the API. On receipt of the input, the API also does some validation before collecting the required data and serving it to the user.

The prototype of the system will serve as a proof of concept thus registration and login of users will not be implemented. However, user registration and login are paramount for the production application. This prevents access to the crowdmap and dashboard by individuals who don't have the right clearance to access the information. The web app also features some machine learning algorithms which should be secured to prevent tampering and unauthorized access.

4.6 Model and Graphical User Interface Design

4.6.1 Filtering Model

The first model we developed and trained was the filtering model. The purpose of the model is to identify whether a tweet is disaster-related or not. If the tweet is classified as irrelevant, it is discarded. If a tweet is disaster-related, i.e. relevant, it's passed onto the next stage of the machine learning pipeline.

Data Collection and Preprocessing

Data used to train the model was collected from CrisisNLP and CrisisLex repositories. Due to our objectives, we employed supervised learning to train the model. We needed to collect only labelled data as we did not have the resources to annotate the tweets manually. The tweets collected were stored in their raw format and had undergone very little preprocessing, if any. We created a preprocessing pipeline that involved dropping irrelevant features, preprocessing the text from the tweets, tokenizing the tweets and embedding them before passing them to the model.

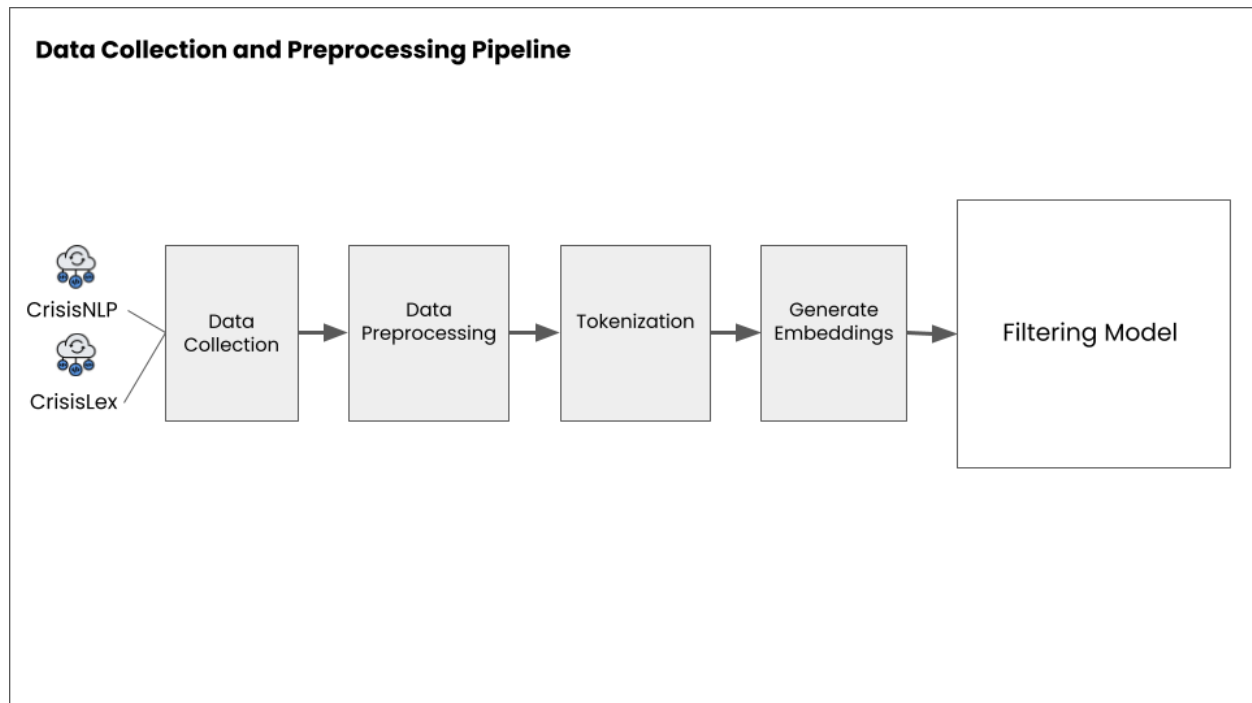


Fig 5. Data collection and preprocessing pipeline for the filtering model

Loaded tweets were in their raw format, hence preprocessing was needed to ensure they were clean to feed to the model. We used the NLTK library and python regex module (re) to preprocess the tweets. We started by defining common contractions in the English language and their expanded meanings. We then created steps to remove hyperlinks, usernames, hashtag symbol, punctuation, stopwords, emojis and the retweet symbol (RT) from the tweets. We achieved this using the regex module. To remove the stopwords, we used the library of stopwords provided by the NLTK library. We also converted the tweets to lowercase and created steps to expand contracted words and lemmatize the words. For the lemmatization, we used the WordNetLemmatizer from the NLTK library.

```
[ ] # Define some common contractions in English
contractions = {
    "ain't": "am not",
    "aren't": "are not",
    "can't": "cannot",
    "can't've": "cannot have",
    "'cause": "because",
    "could've": "could have",
    "couldn't": "could not",
    "couldn't've": "could not have",
    "didn't": "did not",
    "doesn't": "does not",
    "don't": "do not",
    "hadn't": "had not",
```

Fig 6. Some of the word contractions and their expanded equivalents

```
# Remove punctuation marks, stopwords, emojis, urls, convert to lowercase, expand contractions, hashtags, retweet
def preprocess_tweet(tweet):
    res_tweet = []
    lemmatizer = WordNetLemmatizer()

    for word in tweet.split():
        # Expand Contractions
        word = contractions.get(word.lower(), word)

        # Remove stopwords
        if word not in STOPWORDS:

            # Remove url
            word = re.sub(r'http\S+', '', word)

            # Remove usernames
            word = re.sub('@[\w]+', '', word)

            # Remove hashtag
            word = re.sub("#([\w-9A-Za-z \t])|(\w+://\S+)", '', word) # Remove hashtag symbol

            emoji_clean= re.compile("[
                u"\U0001F600-\U0001F64F" # emoticons
                u"\U0001F300-\U0001F5FF" # symbols & pictographs
                u"\U0001F680-\U0001F6FF" # transport & map symbols
                u"\U0001F1E0-\U0001F1FF" # flags (iOS)
                u"\U00002702-\U000027B0"
                u"\U000024C2-\U0001F251"
            ]+", flags=re.UNICODE)
            word = emoji_clean.sub('', word)

            # Remove punctuation
            word = re.sub(r'[^\w\s]', '', word)

            # Remove Retweet
            word = re.sub(r'RT', '', word)

            # Convert to lowercase
            word = word.lower()

            # Lemmatize the word
            word = lemmatizer.lemmatize(word, pos='v')

            if word != '':
                res_tweet.append(word)

    return ' '.join([word for word in res_tweet])
```

Fig 7. Preprocessing steps applied to the tweets

Tokenization

After preprocessing, we needed to tokenize the tweets, create a vocabulary of our data and generate sequences. We used the TensorFlow Tokenizer object to create a dictionary, tokenize, and generate sequences for this step. We defined a maximum sequence length for our tweets to ensure our tweets have a similar number of sequences and our data has a defined shape. Tweets with longer sequences than our specified length were truncated and padded those with shorter sequences with zeros. Tweets have a limited number of characters, currently 280 characters per tweet. After preprocessing of the tweets, the length of the tweets reduced. Thus, we settled for our max sequence length to be 50 sequences as this was a good average to cater for both longer and shorter sequences and gave better accuracy.

```
[ ] # Concatenate our train and test sets, tokenize the tweets and create a vocabulary of our corpus
def prepare_model_input(tokenizer, embedding_dict, X_train, X_test=None, max_sequence_len=50):
    # Combine our training and testing sets
    if X_test is not None:
        text = np.concatenate((X_train, X_test), axis=0)
        text = np.array(text)
    else:
        text = X_train

    tokenizer.fit_on_texts(text)
    sequences = tokenizer.texts_to_sequences(text)
    word_index = tokenizer.word_index # our vocabulary
    text = pad_sequences(sequences, maxlen=max_sequence_len) # max_sequence_len = 50
```

Fig 8. Steps to tokenize, generate a word_index vocabulary and generate sequences

For model architectures that include a pretrained BERT layer, we used the same tokenizer object that was used when training the model to generate tokens and sequences. Tokens were fed into the model without the generation of embeddings afterwards.

Generate Embeddings

Word embeddings allow the representation of words as n-dimensional vectors, which encode the meaning of the word. When plotted, words with similar meaning appear closer in the n-dimensional space. This allows for easier capturing of context and relationship between words. We used GloVe (Global Vectors) word embeddings to represent our tweets vocabulary as

embeddings. We used the glove.6B.100d GloVe embeddings to map our vocabulary to 100-dimensional vectors. We stacked the vectors on top of each other to create a weight matrix to use in our training. This will facilitate faster embedding lookup using the text sequences to obtain the required embedding during the training process.

```
[ ] # Create an embeddings dictionary from the GloVe vectors
def createGloveEmbeddings():
    embeddings_dict = {}
    f = open(GLOVE_DIR, encoding="utf8")
    for line in f:
        values = line.split()
        word = values[0]
        try:
            coefs = np.asarray(values[1:], dtype='float32')
        except:
            pass
        embeddings_dict[word] = coefs
    f.close()
    return embeddings_dict
```

Fig 9. Creating a dictionary of GloVe embedding vectors

```
[ ] # Build weight matrix from the vocabulary and embeddings dictionary
def buildWeightMatrix(word_index, embeddings_dict, EMBEDDING_DIM=100):
    embedding_matrix = np.random.random((len(word_index) + 1, EMBEDDING_DIM))
    for word, i in word_index.items():
        embedding_vector = embeddings_dict.get(word)
        if embedding_vector is not None:
            if len(embedding_matrix[i]) != len(embedding_vector):
                print("could not broadcast input array from shape")
                exit(1)
            embedding_matrix[i] = embedding_vector
    return embedding_matrix
```

Fig 10. Building a weight matrix from the word_index vocabulary and GloVe embedding vocabulary

Model Architecture

We created, trained and evaluated 4 different model architectures for the filtering model. Below is a description of the model architectures:

- a) CNN Model - a model comprising GloVe embeddings, a CNN architecture and fully connected dense layer for classification.

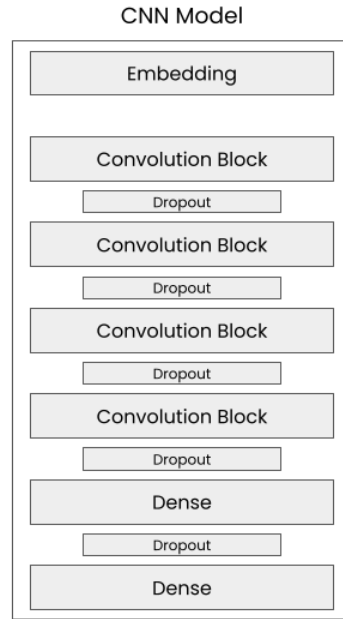


Fig 11. CNN model architecture for the filtering model

- b) BiLSTM model - a model comprising GloVe embeddings, bidirectional LSTM (BiLSTM) layers followed by fully connected dense layers.

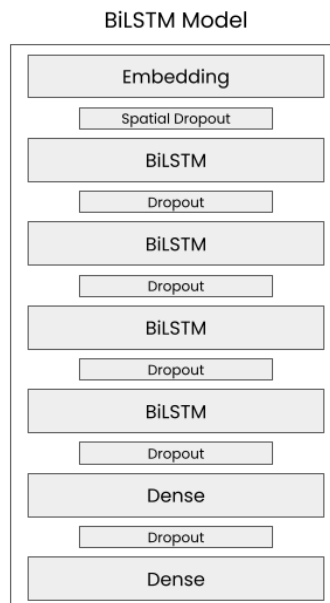


Fig 12. BiLSTM model architecture for the filtering model

- c) BERT_LSTM_ATTENTION - a model comprising a distilBERT layer, followed by BiLSTM layers, an Attention layer and fully connected dense layers. The distilBERT layer parameters are untrainable during the training process.

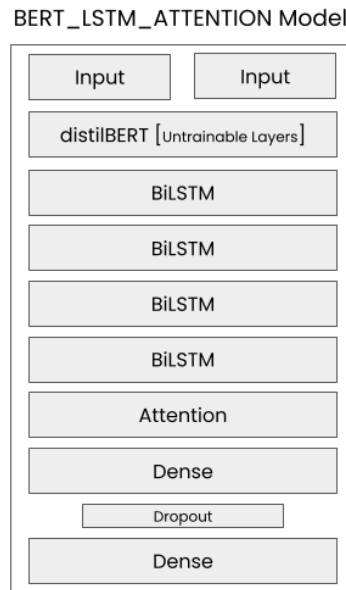


Fig 13. BERT_LSTM_ATTENTION model architecture for the filtering model

- d) BERT_LSTM_ATTENTION_Tuning - a model comprising a distilBERT layer, followed by BiLSTM layers, an Attention layer and fully connected dense layers. In this model, we tune the distilBERT layer by allowing its parameters to be trainable

BERT_LSTM_ATTENTION_Tuning Model

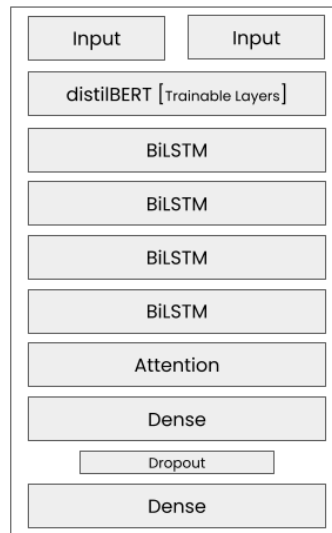


Fig 14. BERT_LSTM_ATTENTION_Tuning model architecture for the filtering model

Training the Model

To train the models, we split our data into 3 main sets, the train, test and validation sets. We used the train set to train the different model architectures and the test and validation sets to evaluate performance. To avoid overfitting, we defined an early stopping callback which monitors the validation loss with a patience of 3 epochs. A consistent increase in the validation loss for 3 consecutive epochs will stop the training process.

4.6.2 Categorizing model

The categorizing model was developed and trained after the filtering model was completed. This model aims to categorize tweets into 5 different humanitarian response activities categories based on the messages of the tweets.

Data Collection and Preprocessing

Data used to train the model was collected from CrisisNLP and CrisisLex repositories. We employed supervised learning to train the model, which meant we had to collect labelled data for our training process. The datasets collected tweets about multiple disasters that happened over some time period and were annotated either by volunteers or paid crowd workers. Since we were building a model to categorize English tweets, we needed to ensure that all tweets in our dataset were in English. Some datasets contained tweets in multiple languages besides English, and those tweets needed to be translated to English before any preprocessing could be attempted. We used the Google Translate API to detect the language of a tweet and translate it to English.

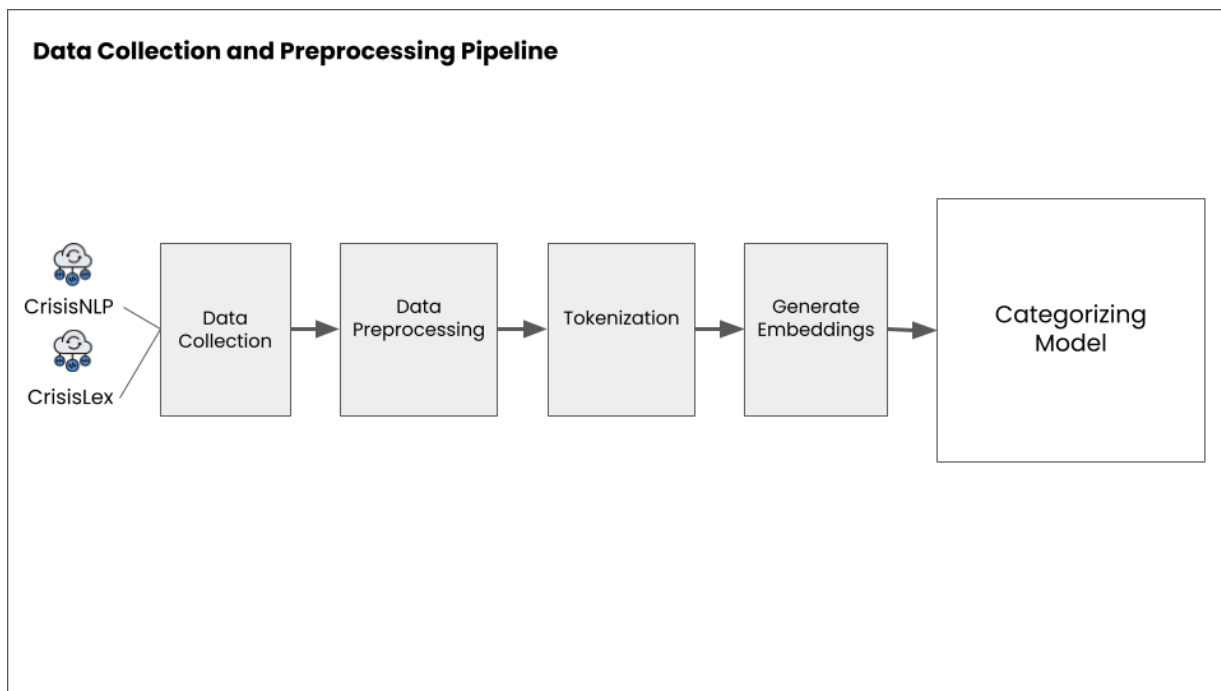


Fig 15. Data collection and preprocessing pipeline for the categorizing model

```

Chile_Earthquake_2014_c1 = pd.read_csv(os.path.join(base_crowd_data_dir, '2014_Chile_Earthquake_c1/2014_Chile_Earthquake_c1_C
Chile_Earthquake_2014_c1.head(2)
]:

```

	tweet_id	tweet_text	label	Lang	Translated Tweet
0	451867084658442240'	Consejos a tener en cuenta si ocurre un terrem...	caution_and_advice	es	Tips to consider if an earthquake like Chile #...
1	451252417024499712'	#Tu_Pornografic Gobierno de Chile confirma cin...	injured_or_dead_people	es	#Tu_Pornografic Government of Chile confirmed ...

```

print("Original Tweet:")
print(Chile_Earthquake_2014_c1.iloc[0]['tweet_text'])
print("Translated Tweet:")
print(Chile_Earthquake_2014_c1.iloc[0]['Translated Tweet'])

print("Original Tweet:")
print(Chile_Earthquake_2014_c1.iloc[2]['tweet_text'])
print("Translated Tweet:")
print(Chile_Earthquake_2014_c1.iloc[2]['Translated Tweet'])

Original Tweet:
Consejos a tener en cuenta si ocurre un terremoto como el de Chile #Mediamza http://t.co/1iKC8QFgh3
Translated Tweet:
Tips to consider if an earthquake like Chile #Mediamza http://t.co/1iKC8QFgh3

Original Tweet:
Terremoto en #Chile y alerta Tsunami: 6 muertos... admirable ejemplo de prevención y disciplina http://t.co/ol0VGijR4I
Translated Tweet:
#Chile earthquake and Tsunami warning: 6 dead ... admirable example of prevention and discipline http://t.co/ol0VGijR4I

```

Fig 16. Sample tweets with their Google Translate translation

We retrieved our features of interest to develop and train, i.e. the tweet id, tweet text and labels from each dataset, then merged the datasets. The combined dataset had over 71000 tweets.

Most of the data from the datasets we collected did not classify tweets with the same labels that we had defined from our research. Thus, we needed to manually create a mapping of all the labels from the datasets that we collected to the 5 categories we defined for use in our label. We obtained all the labels in our merged dataset and created a dictionary mapping each label with the corresponding category. We included an extra category to cater for labels that were not of interest to us or did not have an appropriate category mapping. This allowed us to easily discard tweets that were assigned that category. More than 50% of the tweets had irrelevant categories; hence were discarded.

We put the tweets through the same preprocessing pipeline we had put tweets in the filtering model, i.e. expanding contractions, removing stopwords, links, username, hashtag symbols, emojis, punctuations, retweet symbol, converting them to lowercase and lemmatizing the words in the tweets.

Tokenization and Generating Embeddings

We employed similar techniques to tokenize, generate sequences and pad them, generate word embeddings and create a weight matrix like those used in developing and training the filtering model. The TensorFlow tokenizer object was used to tokenize and create a word-index vocabulary. GloVe embeddings were used to generate embeddings for our tokens and create a weight matrix. We used the same tokenizer object that was used when training the BERT model for any architectures that included a BERT layer. Tokens were fed into the model without the generation of embeddings afterwards.

Model Architecture

We created, trained and evaluated 5 different model architectures for the categorizing model. Below is a description of the model architectures:

- a) Base Model - a model comprising 100-dimensional GloVe embeddings, 4 bidirectional LSTM (BiLSTM) layers followed by fully connected dense layers.

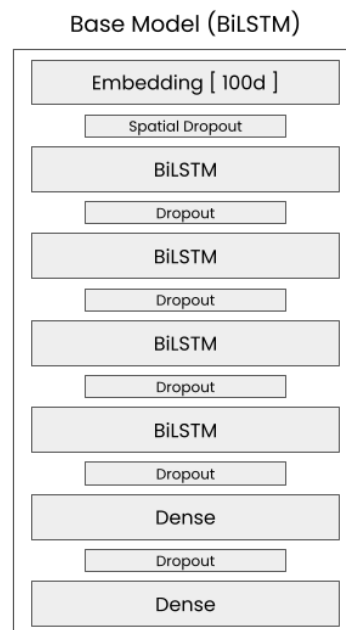


Fig 17. Base model for the categorizing model. Uses 100-dimensional embedding vectors

- b) BiLSTM (200d) model -a model comprising 200-dimensional GloVe embeddings, 4 bidirectional LSTM (BiLSTM) layers followed by fully connected dense layers.

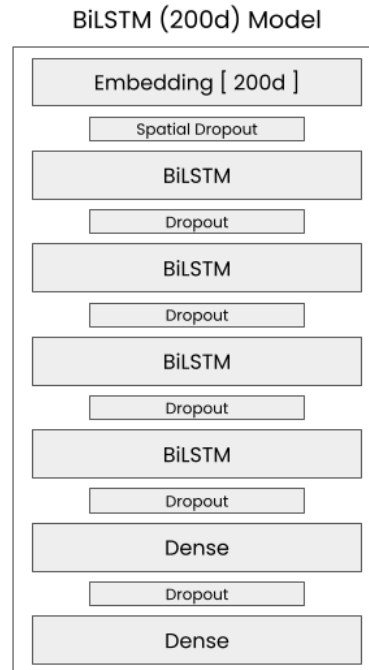


Fig 18. BiLSTM (200d) model architecture for the categorizing model. Uses 200-dimensional embedding

- c) BERT_LSTM_2_Attention- a model comprising a distilBERT layer, followed by 2 BiLSTM layers, an Attention layer and fully connected dense layers. The distilBERT layer parameters are untrainable during the training process.

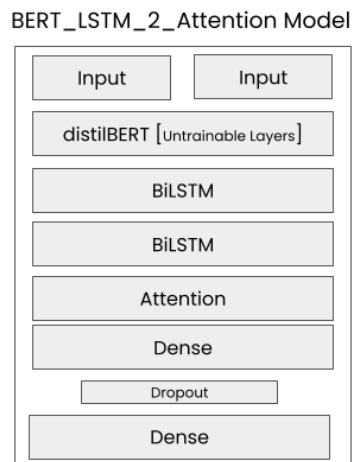


Fig 19. BERT_LSTM_2_Attention model architecture for the categorizing model

- d) BERT_LSTM_4_Attention - a model comprising a distilBERT layer, followed by 4 BiLSTM layers, an Attention layer and fully connected dense layers. The distilBERT layer parameters are untrainable.

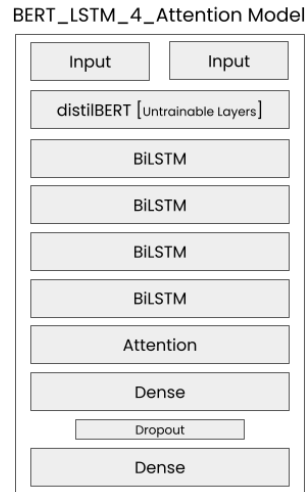


Fig 20. BERT_LSTM_4_Attention model architecture for the categorizing model

- e) BERT_LSTM_ATTENTION_Tuning - a model comprising a distilBERT layer, followed by BiLSTM layers, an Attention layer and fully connected dense layers. In this model, we tune the distilBERT layer by allowing its parameters to be trainable.

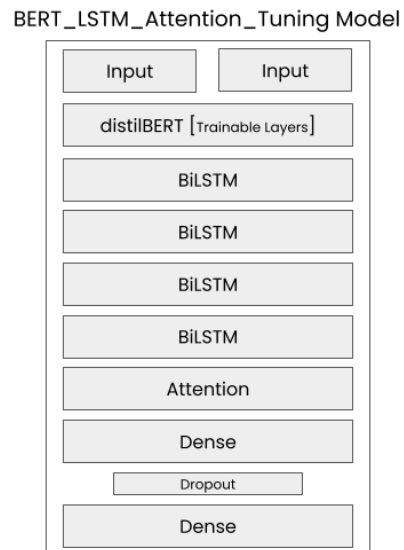


Fig 21. BERT_LSTM_Attention_Tuning model architecture for the categorizing model

Training the Model

Just like when training the filtering model, we split our data into 3 main sets, the train, test and validation sets. We used the train set to train the model architectures and the test and validation sets to evaluate performance. We defined an early stopping with a patience of 3 epochs to avoid overfitting.

4.6.3 Web App Design

Our web app features two main components: a front-end application that the user directly interacts with and a back-end application that acts on behalf of the user and processes information. The system is a web-based dashboard that streams real-time tweets from Twitter using the Twitter Streaming API and creates interactive visualizations that the user can use to get different insights about the tweets.

Front-End

1. Homepage

The front-end of our web app consists of two main pages, the homepage, which collects user input and a dashboard that displays data received from our API using interactive visualizations. Since we want to understand prevailing events during a disaster, we need to collect tweets and analyse them for relevant information. The best way to collect tweets using the Twitter Streaming API for our challenge would be to provide a hashtag or a topic with which we want to stream its tweets. For this, we need to identify hashtags or trending topics that Twitter users are using to share their disaster-related stories and propagate the information. Approximately 1% of tweets on Twitter have geotagged information associated with them [23]. To ensure that we only collect tweets with geotag information related to our hashtag or topic of interest, we need to provide a location filter to the streamed tweets. Thus, we included an option to submit a country of interest from which you want the tweets. On the homepage, we have a form that collects two primary information from the user that we use to stream tweets: hashtag(s) or topic(s) and a country. On submit, the front-

end application makes a post request to our API, which returns a success message if the request was completed successfully.

On receipt of a success message, the app redirects to the dashboard page and sends a GET request to the API to retrieve any streamed tweets. The app sends a GET request to the API every 10 seconds to collect newly streamed tweets and ensure the data on the app is real-time data.

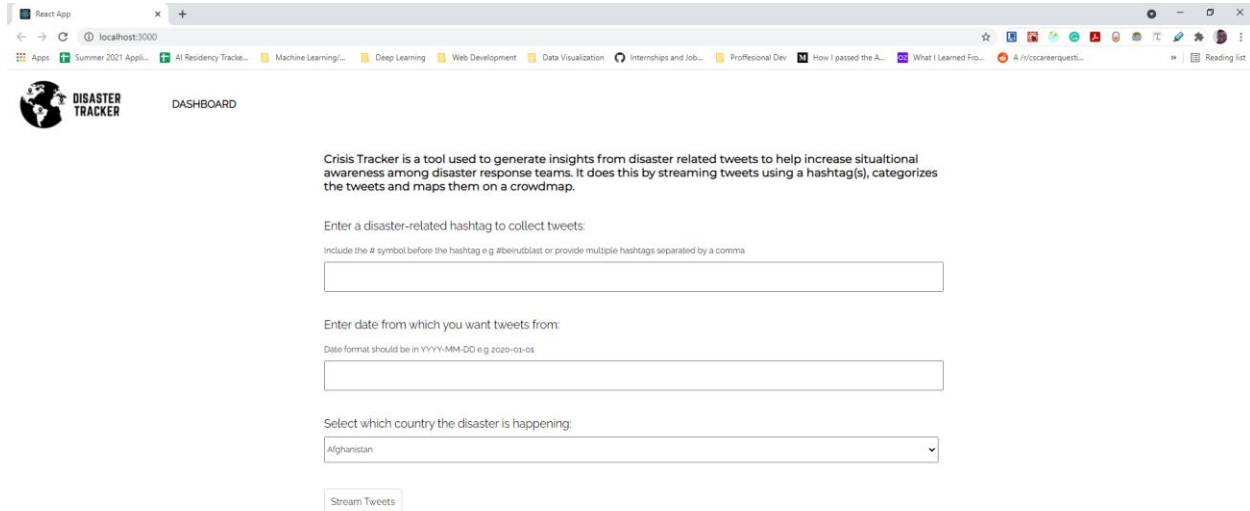


Fig 22. The homepage of the application features a form to collect information to stream the tweets.

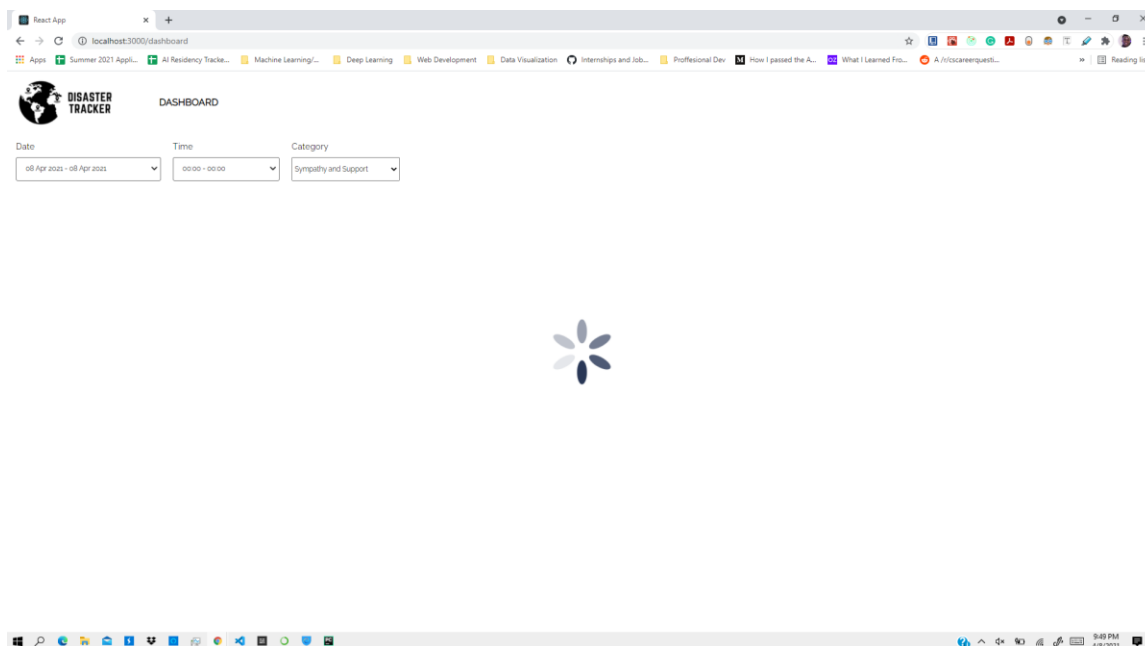


Fig 23. A loading icon is displayed as the app fetches data from the API

2. Dashboard

The dashboard is an interactive tool that features a crowd map, distribution plots of several properties of the tweets, and a tweet list. The crowdmap plots each tweet on a map using its coordinates and identifies the categories of the tweets using different colors. The crowdmap, which is built on Google Maps, allows you to drill down to the exact location of a tweet and identify landmarks, buildings and the surrounding area from which the tweet was shared from. The dashboard also features a bar plot that shows the distribution of the tweets in each of the 6 categories that the tweets were classified. A list of the tweets is also available to provide more information about the tweets as they stream in. The dashboard has several page-level filters that filter all the data on the page. The first is a date filter that allows filtering tweets to those within a specific set of dates, a time filter to filter tweets shared within a certain time frame and a category filter to filter tweets by their predicted categories.

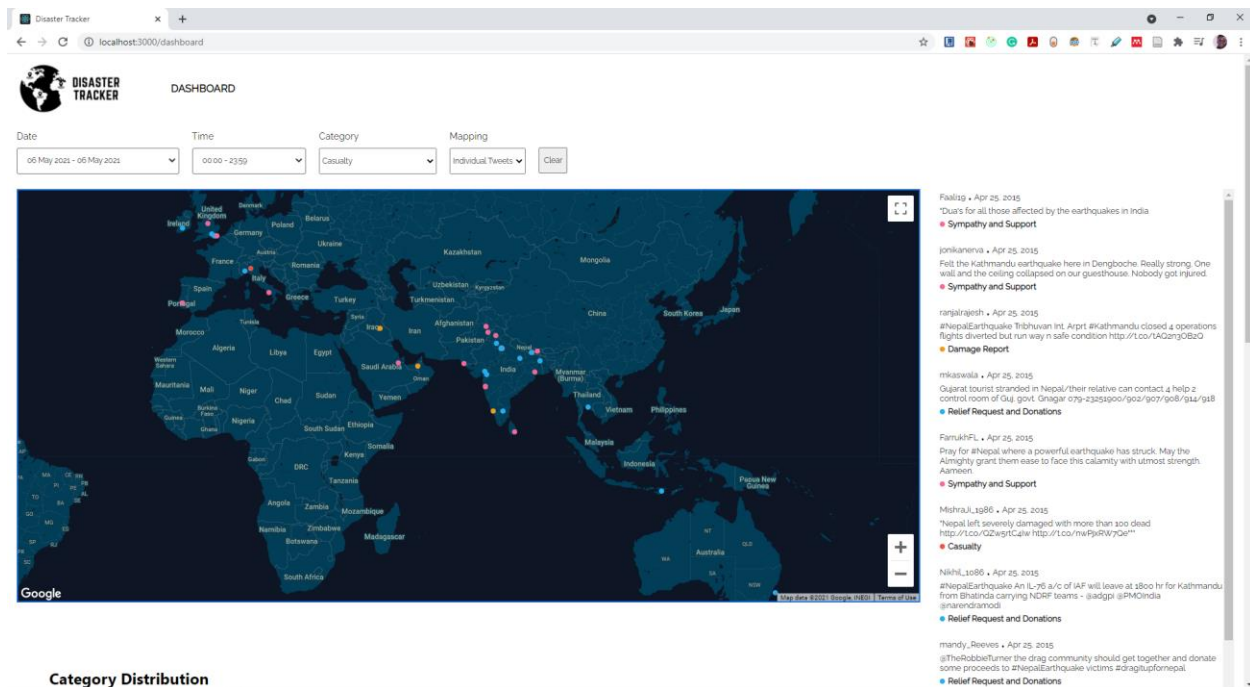


Fig 24. The dashboard page showing the crowdmap and a section of the tweets list



Fig 25. Zoomed in image of the crowdmap

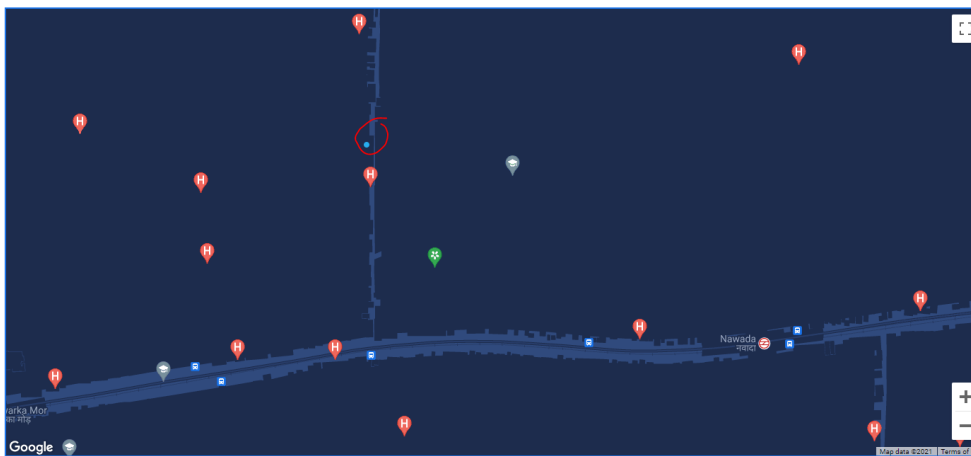


Fig 26. Zoomed in image of a tweet location showing the surrounding area, landmarks and roads

- ranjalrajesh • Apr 25, 2015
#NepalEarthquake Tribhuvan Int. Arprt #Kathmandu closed 4 operations
flights diverted but run way n safe condition <http://t.co/1AQ2ngOB2Q>
● **Damage Report**
- mkaswala • Apr 25, 2015
Gujarat tourist stranded in Nepal/their relative can contact 4 help 2
control room of Guj. govt. Gnagar 079-23251900/902/907/908/914/918
● **Relief Request and Donations**
- FarrukhFL • Apr 25, 2015
Pray for #Nepal where a powerful earthquake has struck. May the
Almighty grant them ease to face this calamity with utmost strength.
Aameen.
● **Sympathy and Support**
- MishraJi_1986 • Apr 25, 2015
"Nepal left severely damaged with more than 100 dead
<http://t.co/OZw5rtC4lw> <http://t.co/nwPjxRW7Qe>"
● **Casualty**

Fig 27. Tweets list section. Tweets are displayed with the author, date, tweet text and predicted category.

Category Distribution

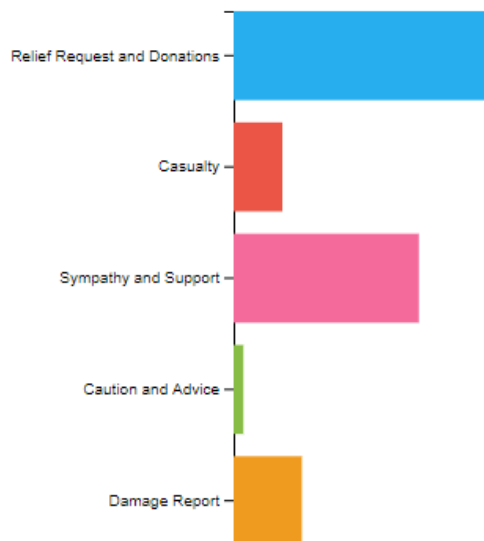


Fig 28. Image showing the category distribution of tweets.

3. Architecture

The front-end of our web app was entirely built on React.js, a JavaScript framework that allows the creation of reusable components. We used Redux to manage the state of our application. Redux enables the creation of a global state, the store, from which you store the state of the application and data to be rendered. Our redux store kept track of our tweets and passed the data to each of the individual components. Actions lead to data creation, which updates the store and results in all the components getting data from the store re-rendering. When the application makes a GET request to the API, the response retrieved is used to update the store with the most recent data. An update of the store causes all other components, i.e. the crowdmap, tweets list and category distribution plot to re-render and visualize the new data that has been updated. This results in a dynamic app that maps tweets and visualizes tweets' information as they are streamed.

D3.js is used to create dynamic and interactive visualization on the dashboard. The category distribution is created using D3.js and displays information about a category when the mouse hovers over the bars.

The crowdmap is built on Google Maps. With the appropriate credentials, we fetch the map data from the Google Maps API and display our data on the map. The map object provided by the Google Maps API allows for the addition of custom data on top of the data displayed on the map. We use this functionality to map out the location of the tweets using their coordinates and plot them. Since we are using GPS coordinates, we can plot the exact location from which a tweet was shared from.

Back-End

For the back-end, we built a REST API using Flask and Python. The API processes user input sent by the front-end application and uses the information to stream tweets from the Twitter Streaming API. It then passes the tweets through a preprocessing pipeline and a machine learning pipeline before converting the tweets to GeoJSON, storing them temporarily and sending them to the front-end application after a GET request.

1. Request processing

The API has a single endpoint that can be accessed using two HTTP methods, the GET method and the POST method. The first API call from the front-end application will be using the POST method. This is when the user submits their input for the hashtag(s) or topic(s) and country information. This information is retrieved from the body of the request object. A new thread is created and initialized with the information obtained from the request object. This thread is responsible for streaming tweets from the Twitter Streaming API, processing them and storing them temporarily even after the main thread stops execution after sending a response to the front-end application. The new thread is then started, and a success message is sent to the front-end application to indicate successful submission of information and initialization of the streaming process. The main thread then stops execution.

```

@app.route('/tweets-query', methods=["POST"])
@cross_origin(supports_credentials=True)
def postInput():
    if request.json is None:
        return "ERROR: No query was provided. Kindly include the required query parameters"
    else:
        hashtag = request.json['hashtag'] if 'hashtag' in request.json else None
        date = request.json['date'] if 'date' in request.json else None
        country = request.json['country'] if 'country' in request.json else None

        if hashtag and date and country:
            hashtag = [tag.strip() for tag in hashtag.split(",")]

            t = Thread(hashtag, date, country)
            t.start()

            return {"Status_Code": 200,
                    "Status": "Success"}

        else:
            return 'ERROR: Wrong queries provided'

```

Fig 29. Processing of request object body information and initialization of our custom thread object

The next API call will be after the successful submission and initialization of the streaming thread. After the front-end receives the success message, it sends multiple GET requests to the same API endpoint every 10 seconds. A GET request returns data that has been temporarily stored in the API by the streaming thread.

2. Tweets Streaming

After the new thread has been created, the API retrieves Twitter API authentication tokens from file and uses them for authentication on Twitter. We use the Tweepy library to facilitate authentication on the Twitter API and streaming of tweets.

After successful authentication, we define a stream listener. A stream listener is an object that defines how to handle tweets after they are streamed. We inherit from the Tweepy StreamListener, which provides us access to override its methods for handling tweets. When a tweet has been streamed from the Twitter Streaming API, we first check if the tweet has geolocation information associated with it, i.e. we check whether the tweet's coordinates object has a value. If so, we extract key tweet information relevant to us, e.g. tweet id, tweet text, user screen name, the tweet's created

date, tweet's coordinates and hashtags, among others. We create a dataframe of the tweet to easily structure the tweet's information. We then pass the tweet's text through a preprocessing pipeline that expands contractions, removes stopwords, links, usernames, hashtag symbols, emojis, punctuation, retweet symbol, converts the tweet to lowercase and lemmatizes each word in the tweet. We use Python's re module for regex operations and the NLTK library. This pipeline is similar to the preprocessing pipeline used when preprocessing tweets before training the machine learning models.

We then pass the tweet through the filtering model. This model identifies the relevance of the tweet, i.e. if the tweet is disaster-related or not. If not, the tweet is discarded and not stored. If the tweet is identified as relevant, it is fed to the categorizing model, which categorizes the tweet into one of 6 categories and maps its numeric category output to a string category.

We convert the tweet's collected information into a GeoJSON object before storing it to a list. A GeoJSON object provides a format that allows us to easily store geographic information together with non-geographic information for easy mapping. Geographic information is stored in the geometry object, and all other attributes in the properties object. We store the tweet's coordinates as a list in the geometry feature and other relevant information in the properties object for our tweet. We then add the object in a list that acts as temporary storage of data in our API.

With our stream listener defined, we instantiate the tweet stream with our Twitter authentication object and the stream listener. We filter the collected tweets by the hashtag(s) or topic(s) and country. For the country, we create a dictionary of countries and their bounding boxes. These are the geographic coordinates that define the location of a country on a map. We retrieve the appropriate bounding box and pass it alongside the hashtags(s) in the filter.

```

tweet_data = {}
geometry = {}
properties = {}

tweet_data['type'] = "Feature"
tweet_data['geometry'] = geometry
tweet_data['geometry']['type'] = "Point"
tweet_data['geometry']['coordinates'] = [float(geo_coordinates_lng), float(geo_coordinates_lat)]
tweet_data['properties'] = properties
tweet_data['properties']['tweet_id'] = tweet_id
tweet_data['properties']['tweet_text'] = tweet_text
tweet_data['properties']['user_location'] = user_location
tweet_data['properties']['user_screen_name'] = user_screen_name
tweet_data['properties']['created_at'] = created_at
tweet_data['properties']['place_name'] = place_name
tweet_data['properties']['bounding_coord_1'] = [float(coord) for coord in bounding_coord_1]
tweet_data['properties']['bounding_coord_2'] = [float(coord) for coord in bounding_coord_2]
tweet_data['properties']['bounding_coord_3'] = [float(coord) for coord in bounding_coord_3]
tweet_data['properties']['bounding_coord_4'] = [float(coord) for coord in bounding_coord_4]
tweet_data['properties']['category'] = tweet_categories[0]
tweet_data['properties']['hashtags'] = hashtags
tweet_data['properties']['cluster'] = False

```

Fig 30. Creation of the GeoJSON object using the tweet's information

```

'AF': ('Afghanistan', (60.5284298033, 29.318572496, 75.1580277851, 38.4862816432)),
'AO': ('Angola', (11.6400960629, -17.9306364885, 24.0799052263, -4.43802336998)),
'AL': ('Albania', (19.3044861183, 39.624997667, 21.0200403175, 42.6882473822)),
'AE': ('United Arab Emirates', (51.5795186705, 22.4969475367, 56.3968473651, 26.055464179)),
'AR': ('Argentina', (-73.4154357571, -55.25, -53.628348965, -21.8323104794)),
'AM': ('Armenia', (43.5827458026, 38.7412014837, 46.5057198423, 41.2481285671)),
'AQ': ('Antarctica', (-180.0, -90.0, 180.0, -63.2706604895)),
'TF': ('Fr. S. and Antarctic Lands', (68.72, -49.775, 70.56, -48.625)),
'AU': ('Australia', (113.338953078, -43.6345972634, 153.569469029, -10.6681857235)),
'AT': ('Austria', (9.47996951665, 46.4318173285, 16.9796667823, 49.0390742051)),
'AZ': ('Azerbaijan', (44.7939896991, 38.2703775091, 50.3928210793, 41.8606751572)),
'BI': ('Burundi', (29.0249263852, -4.49998341229, 30.752262811, -2.34848683025)),

```

Fig 31. A snippet of the bounding boxes for all countries

Chapter 5: Implementation and Testing

5.1 Introduction

In this section, we'll present the results from testing our system. We'll present results obtained from our unit tests and results from our integration testing. Our units consisted of the different models we trained and evaluated, and the modules comprising our deployed application. The integration tests section will explore results from integrating our system modules and their performance on different systems and test cases.

5.2 Machine Learning Models Results

To evaluate our models, both the filtering model and the categorizing model, we defined different model architectures for each model and tracked the performance of the models with the same data. The data was split into 3 main sets, the train set used in training the models, the validation set which was used to test model performance during training and estimate its performance, and the test set used to evaluate model performance after models had completed training. The data was split at a 70-10-20 split for the train, validation and test sets, respectively.

Since our models were both performing classification tasks, classification metrics were used to evaluate their performance. The models were evaluated based on their accuracy, precision, recall, f1-score.

Accuracy is the ratio of predictions the model predicted correctly over the total number of predictions. A higher accuracy score is better.

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Precision is the ratio of true positives (observations correctly classified as belonging to a class) and the total number of observations labelled as belonging to the positive class.

$$Precision = \frac{TP}{TP + FP},$$

where TP, True Positives, is the number of observations correctly classified as. positive i.e. belonging to a class and FP, False Positives, is the number of observations classified as positive while they are labelled as negative i.e. not belonging to that particular class. Precision allows us to answer the question: What proportion of positive classifications were correct? A higher precision score is better.

Recall is the ratio of true positives and the number of observations belonging to the particular class. High

$$Recall = \frac{TP}{TP + FN},$$

where FN, False Negatives, is the number of observations that have been classified as false i.e. classified as belonging to a different class, while they belong to this particular class. Recall allows us to answer the question: What proportion of actual positives was identified correctly? A higher recall score is better.

F1-score is the harmonic mean between precision and recall. A higher f1-score is better.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

We also plotted a confusion matrix for each of the models to get a deeper understanding of the model performance with the different categories.

1. Filtering Model Results

Table 2 shows the results obtained from the filtering model. We present the results in a table against the different model architectures that were created and trained. We also plot confusion matrices of the models to look at how the models performed to identify a tweet as relevant or irrelevant and look at which category the models were struggling to identify correctly. We then plot a graph of the results for each model for easier comparison.

Table 2. Accuracy, precision, recall and f1-scores of the different model architectures of the filtering model

Model	Accuracy	Precision	Recall	F1-score
CNN Model	0.77	0.79	0.76	0.76
BiLSTM Model	0.82	0.83	0.82	0.76
BERT_LSTM_ ATTENTION	0.83	0.83	0.83	0.83
BERT_LSTM_ ATTENTION_ Tuning	0.84	0.84	0.84	0.84

Plot of Accuracy, Macro Precision, Macro Recall and Macro F1-score

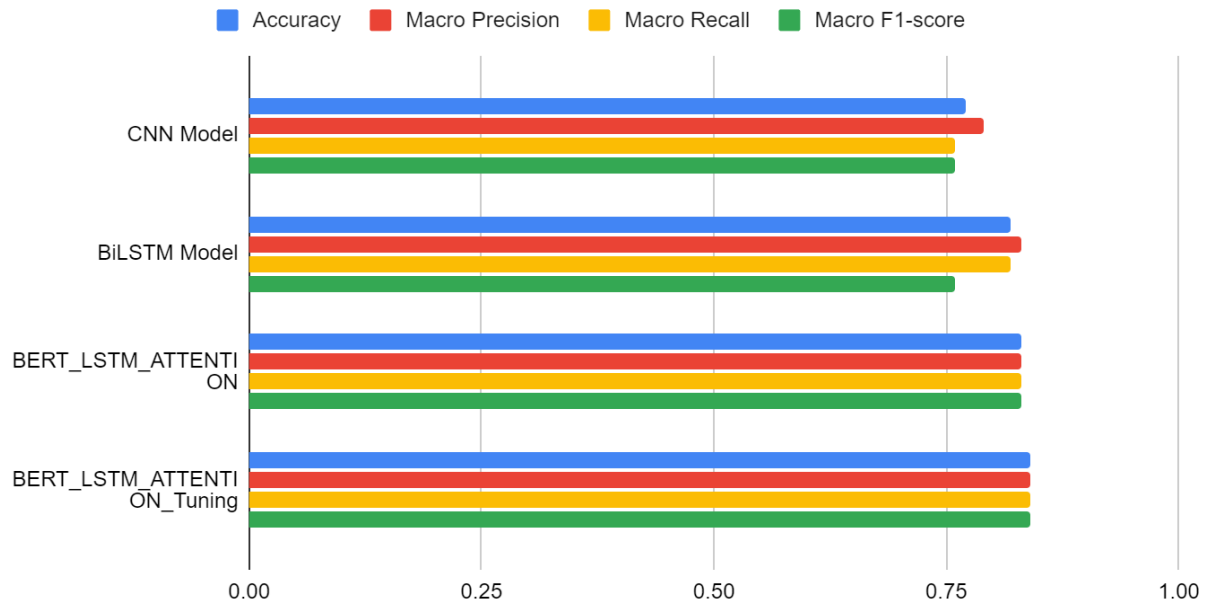


Fig 32. Plot of model accuracy, precision, recall and f1-scores of the different model architectures

From the evaluation results on Table 2 and Fig 32, the BERT (Bidirectional Encoder Representations from Transformers) models outperform all the other model architectures. This could be due to the fact that the BERT model architectures use transfer learning from pre-trained BERT models. These models have been pre-trained on very large corpora and are good at identifying context within a text. For our models, we used a distilBERT model which is a lightweight BERT model that is easy to deploy to applications and faster to train. From the first CNN model that we trained, there was a 7% increase in the performance of the BERT_LSTM_ATTENTION_Tuning. The latter model, despite using transfer learning, tunes its model weights to the current classification task hence is able to achieve higher accuracy than the other untuned BERT models. The model also registers an increase in its recall, precision and f1-score indicating that it has less bias in identifying whether a tweet is relevant or irrelevant compared to other model architectures.

Confusion Matrices

CNN

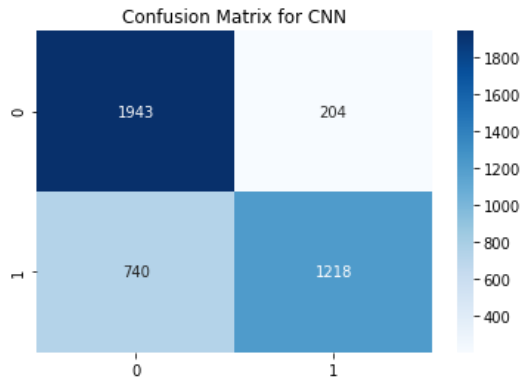


Fig 33. Confusion matrix for the CNN model

BiLSTM

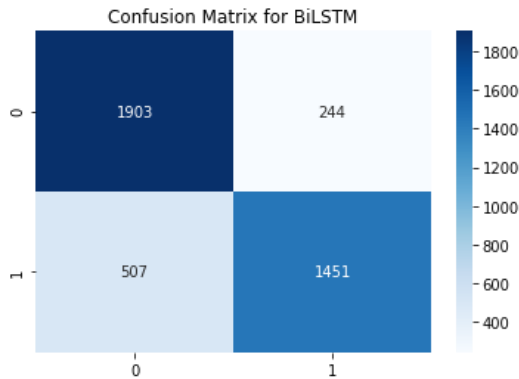
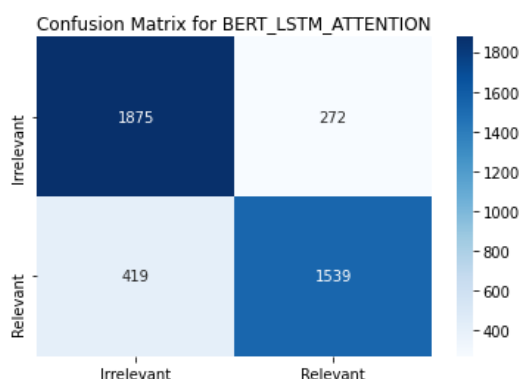
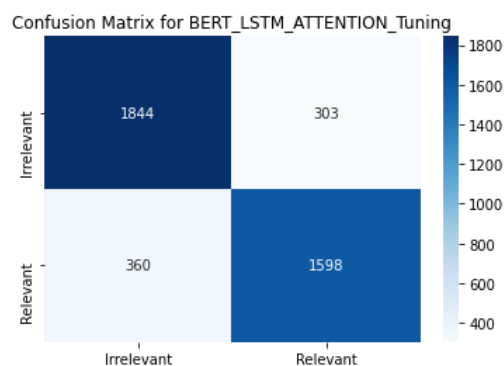


Fig 34. Confusion matrix for the BiLSTM model

BERT_LSTM_ATTENTION

**Fig 35.** Confusion matrix for the BERT_LSTM_ATTENTION model

BERT_LSTM_ATTENTION_Tuning

**Fig 36.** Confusion matrix for the BERT_LSTM_ATTENTION_Tuning model

A deeper look at how the models performed at predicting relevance of the tweets (Fig. 33, Fig. 34, Fig. 35, Fig. 36) shows that the models performed relatively better at identifying irrelevant tweets as opposed to relevant tweets. In the test set, irrelevant tweets had a support (number of samples of a class in a dataset) of 2147, while relevant tweets had a support of 1958. The initial CNN model had a huge bias towards identifying irrelevant tweets correctly with a misclassification of 10% of irrelevant tweets against a misclassification of 38% of relevant tweets. The BiLSTM model got slightly better at identifying relevant tweets only misclassifying 26% of the relevant tweets but performed slightly poorly at identifying irrelevant tweets with a misclassification of 11% of the samples. The BERT_LSTM_ATTENTION model did even better and reduced the bias against relevant tweets. It only misclassified 21% of the tweets while performing slightly poorly at identifying irrelevant tweets with a misclassification of 13% of the tweets. The BERT_LSTM_ATTENTION_Tuning was the best performing model with just a 4% difference between its ability to identify relevant and irrelevant tweets from 28% difference in the CNN model. The model misclassified 14% of irrelevant tweets and only 18% of the relevant tweets. As we used different model architectures, the filtering model was able to reduce its bias towards predicting tweets as irrelevant and got better at identifying relevant tweets.

2. Categorizing Model Results

Table 3 shows the results obtained from evaluating the categorizing model on accuracy, precision, recall and f1-score. We plot a graph of the results for each model for easier comparison. We also plot confusion matrices of the models to look at how the models performed in categorizing the tweets into the 5 humanitarian response activities (Sympathy and Support, Casualty, Caution and Advise, Damage Report, Relief Request and Donations) and have a look at which category the models were struggling to correctly classify.

Table 3. Accuracy, precision, recall and f1-scores of the different model architectures of the categorizing model

Models	Accuracy	Precision	Recall	F1-score
Base Model (BiLSTM)	0.73	0.74	0.74	0.74
BiLSTM (200d)	0.74	0.74	0.74	0.74
BERT_LSTM_ 2_Attention	0.77	0.78	0.78	0.78
BERT_LSTM_ 4_Attention	0.78	0.79	0.79	0.79
BERT_LSTM_ Attention_Tuning	0.84	0.84	0.84	0.84

Plot of Accuracy, Macro Precision, Macro Recall and Macro F1-score

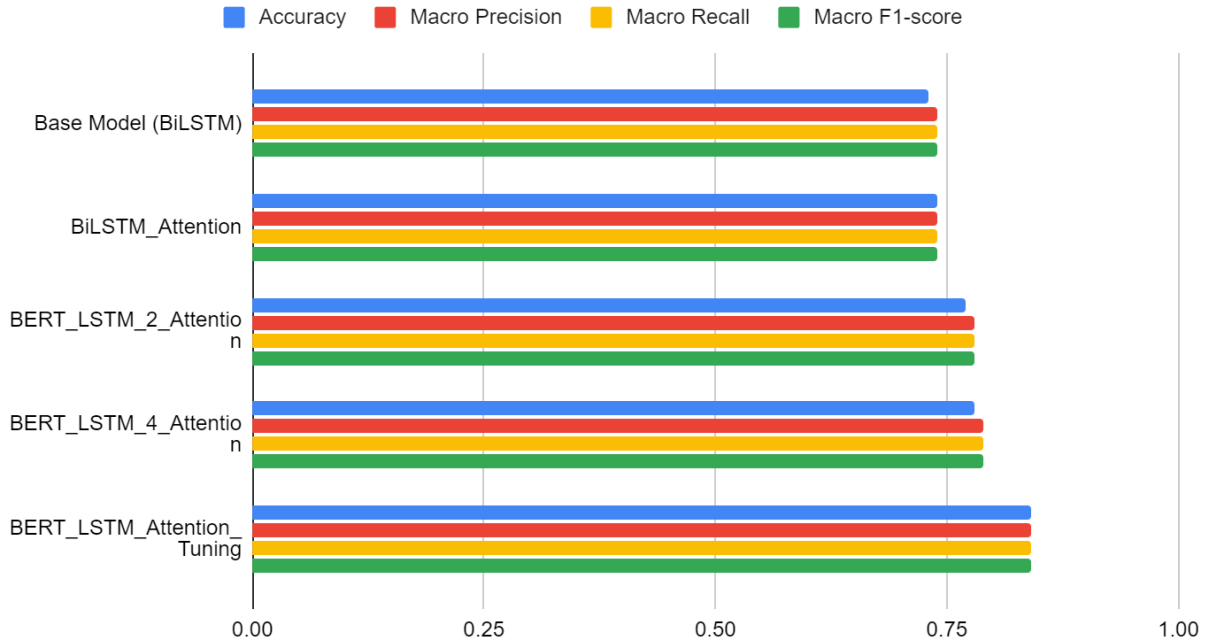


Fig 37. Plot of model accuracy, precision, recall and f1-scores of the different model architectures

From the evaluation results on Table 3 and Fig 37 on the categorizing model, the BERT models outperformed the Base Model and BiLSTM model in all the metrics. As discussed in the filtering model, BERT models are better at identifying context in text hence perform better at categorizing the tweets. As with the filtering model, we used a distilBERT model for all the architectures with a BERT model to allow faster training and easier deployment of the model. The best performing model architecture, the BERT_LSTM_Attention_Tuning features a fine-tuned BERT model with BiLSTM layers, an Attention head and dense layers. The BERT model is fine-tuned, meaning, its layers are unfrozen and its weights get updated in the training process to allow it to better understand the context of our dataset and accurately categorize the tweets. The BERT_LSTM_Attention_Tuning model registered an 11% increase in its accuracy and a 10% increase in its precision, recall and f1-score compared to the Base model. This indicates that the model is much better at understanding the message of the tweets and correctly categorizing them.

Confusion Matrices

Base Model (BiLSTM)

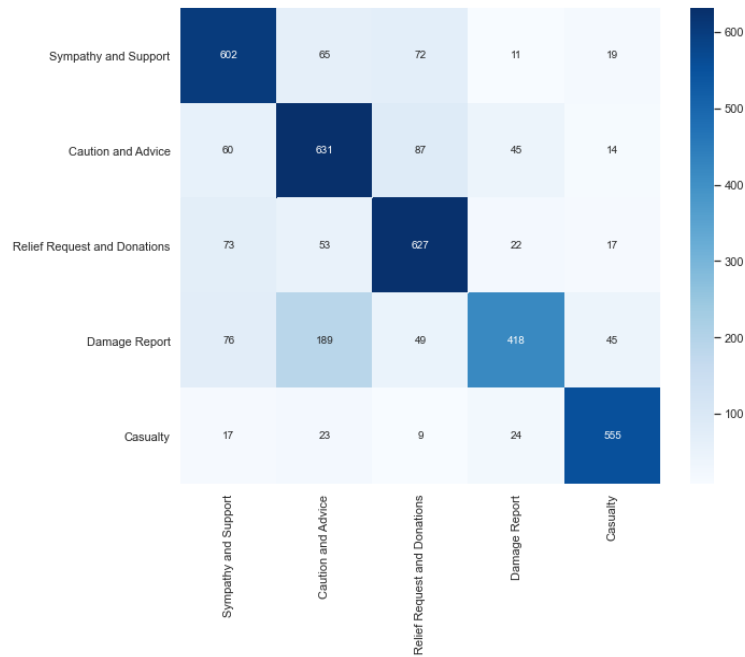


Fig 38. Confusion matrix for the Base Model (BiLSTM)

BiLSTM (200d)

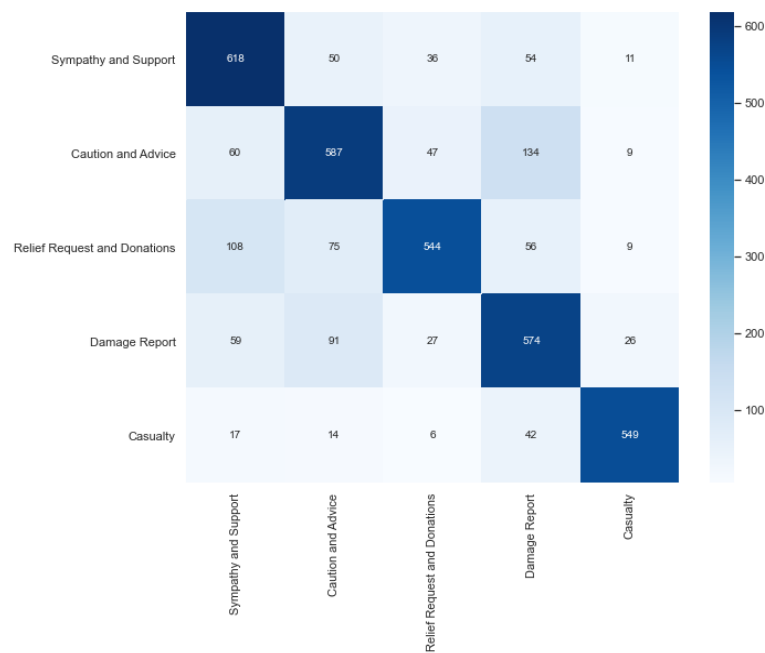


Fig 39. Confusion matrix for the BiLSTM (200d) model

BERT_LSTM_2_Attention

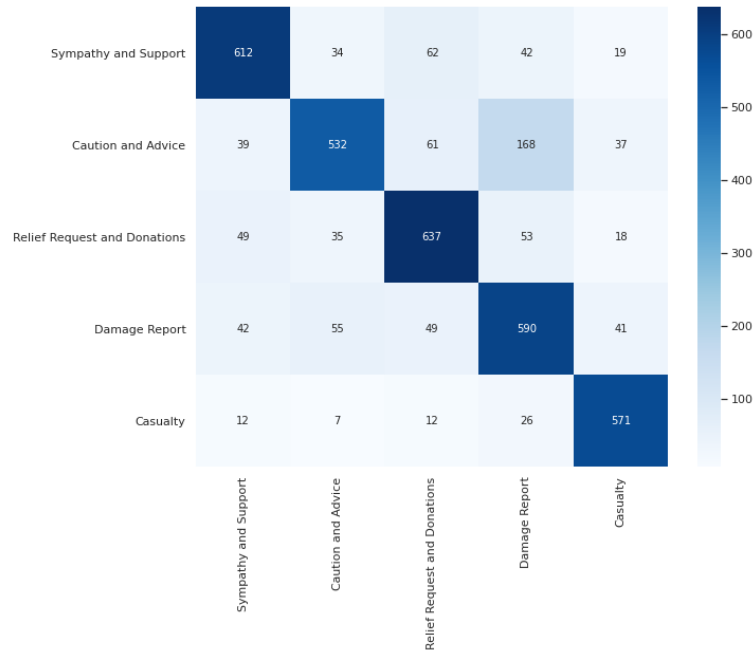


Fig 40. Confusion matrix for the BERT_LSTM_2_Attention model

BERT_LSTM_4_Attention

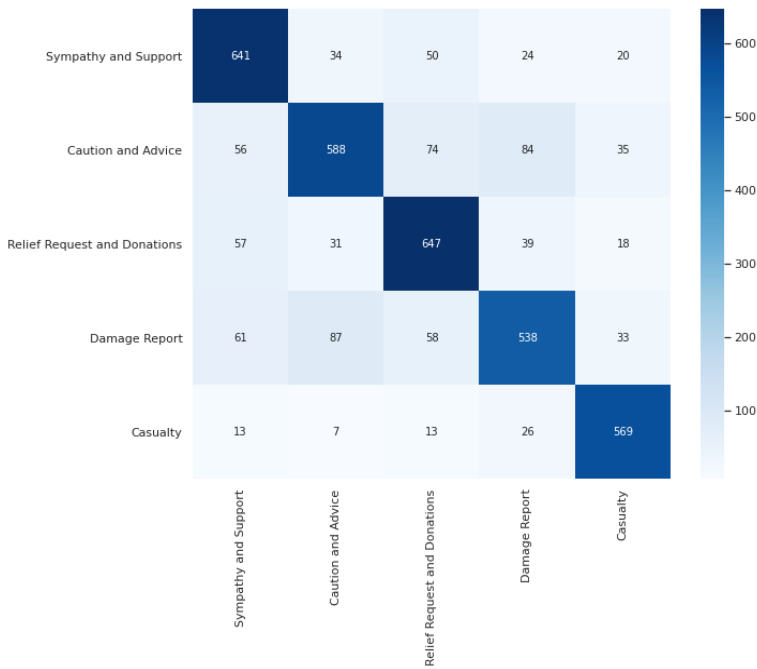


Fig 41. Confusion matrix for the BERT_LSTM_4_Attention model

BERT_LSTM_Attention_Tuning

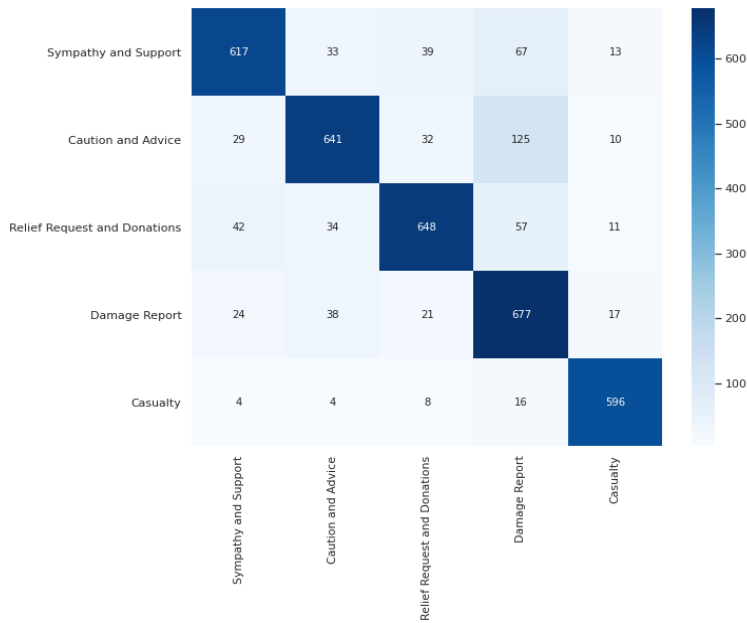


Fig 42. Confusion matrix for the BERT_LSTM_Attention_Tuning model

From the confusion matrices above (Fig. 38 - 42), we observe that as the model complexity increases and its architecture gets more advanced, the better the accuracy of the model gets with each individual class. The models however struggle at distinguishing Caution and Advice tweets and those tagged as Damage Report as it misclassified on average about 13% of the former as the latter. This could be because some of the tweets labelled as Caution and Advice had some information about infrastructure or utilities damage in their message. Despite this, the core message of these tweets was caution and advice and not reporting about damages. In terms of the models' accuracy in predicting each category, the best performing model architecture, BERT_LSTM_Attention_Tuning performed really well at categorizing Casualty tweets with an accuracy of approximately 95%. On the other hand, it performs poorly at categorizing Caution and Advice tweets with an accuracy of approximately 77%.

5.3 System Accuracy

To further test the accuracy of the models after training, we compiled 5 test cases from which we tested the models. We sampled 5 tweets from 5 different disasters (1 from each disaster) that were

collected as part of tweets posted during these disasters with the disasters' hashtags. We tested our best performing model architectures for both the filtering and categorizing models that were later deployed to the web application. For the filtering model, we tested the BERT_LSTM_ATTENTION model while for the categorizing model, we tested the BERT_LSTM_ATTENTION_Tuning model.

Tweet sample 1 (From the 2014 California earthquake):

Napa Earthquake - First A Boom, Then Flames At Mobile Home Park #jobs
<http://t.co/Eu7MvlUCDp> <http://t.co/QvvgbxdeDY>

Filtering Model: Relevant

Categorizing Model: Damage Report

Tweet sample 2 (From the 2015 Nepal Earthquake):

Death toll rises to 870. Makes our heart sink. Emergency declared in Nepal. Pls contribute to PM's Relief Fund

Filtering Model: Relevant

Categorizing Model: Relief Request and Donations (Actual label: Casualty)

Tweet sample 3 (From the 2014 Chile Earthquake):

On a day like this 32 years ago, Argentina illegally invaded the Falklands. Up to this day they don't show repent nor do they admit defeat.

Filtering Model: Irrelevant

Categorizing Model: Casualty

Tweet sample 4 (From the 2014 Philippines Typhoon Hagupit):

Here are some Storm Survival Tips in preparation for #Hagupit #RubyPH| Be Prepared @PNPh hotline @pnppio @PNP_PCRG <http://t.co/e...> .

Filtering Model: Relevant

Categorizing Model: Caution and Advice

Tweet sample 5 (From the 2014 India Floods):

RT @TheWhaleFacts: Whales usually dress up as whales for Halloween.

Filtering Model: Irrelevant

Categorizing Model: Sympathy and Support

In the implementation of the system, tweets identified as irrelevant are not allowed to proceed to other preprocessing steps or to the categorizing model. They are simply discarded. Incorrect classification of these tweets by the categorizing model does not indicate poor performance as the model is forced to assign a category to the tweet despite not being trained on related information and not having an appropriate category to assign it. In the design of the machine learning models pipeline, the model won't be required to do this.

5.4 Functional Testing

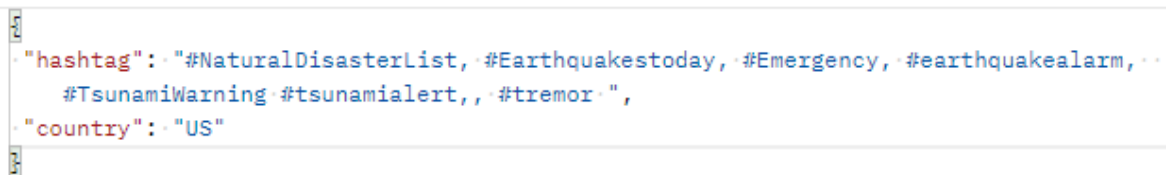
1. Unit Testing

We divided our system into 2 major components to perform unit tests and integration tests. These are:

a) Streaming Module

The streaming module encompasses all units developed for the Disaster Tracker API. Individual functions of the module were tested with sample test cases and their output observed before being integrated with other units. The units were integrated to form the streaming module which we tested to ensure compatibility of the units and data flows correctly between the units to produce the required output. To test the streaming module, we used Postman to test our API by sending requests to the API and observing the response sent. Postman is an API client that allows the creation, sharing and testing of API by sending HTTP or HTTPS requests.

To test, we create a POST request to the API with sample input that is required by the API to process the request. In the request, we add a JSON object with a hashtag and country property in the body and send the request. We then observe to ensure the request was received by the API and wait for it to process it and send a response.



```
{  
  "hashtag": "#NaturalDisasterList, #Earthquakestoday, #Emergency, #earthquakealarm, #TsunamiWarning #tsunamialert, #tremor",  
  "country": "US"  
}
```

Fig 43. Sample input included in the body of the request to the API

We then send a GET request to the API and observe the response that it received.

```
{
  "geometry": {
    "coordinates": [
      -114.16,
      46.25
    ],
    "type": "Point"
  },
  "properties": {
    "bounding_coord_1": [
      -114.18379,
      46.196905
    ],
    "bounding_coord_2": [
      -114.18379,
      46.279435
    ],
    "bounding_coord_3": [
      -114.134782,
      46.279435
    ],
    "bounding_coord_4": [
      -114.134782,
      46.196905
    ],
    "category": "Sympathy and Support",
    "cluster": false,
    "created_at": "Mon Apr 19 13:32:59 +0000 2021",
  }
}
```

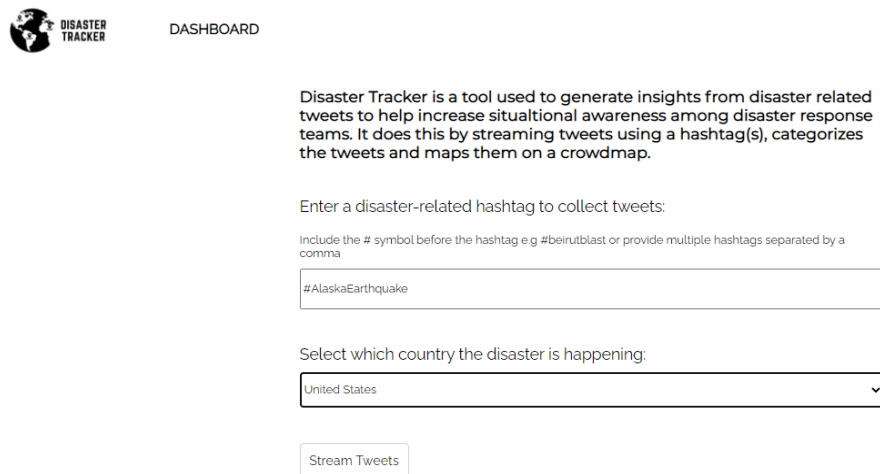
Fig 44. Sample response from the streaming module

The input of the country's value is limited to a dropdown that restricts entering incorrect values. The user is thus given a fixed option list to select the value. The hashtag value is open to whatever type of value a user wishes to input with restrictions on characters. The field receives a string input which if separated by a comma is converted to a list of values. Users can either include the hash symbol (#) to indicate a hashtag or ignore it to indicate a topic. Below are screenshots of how the streaming module responds to different input values from the user.

b) Display Module

The display module is responsible for receiving user input and displaying tweets to the user on the front-end of the system. The displayed data depends on the data received from the streaming module, which depends on the user input. Thus, ensuring correct user input was paramount to the

correct working of the system. Most of the user input in this module is restricted to a specified list of options to allow easy processing of the information. The only non-option field that collects user input is the hashtag field. The user is required to input a string or a list of strings separated by a comma. Users can either include the hash symbol (#) to indicate a hashtag or ignore it to indicate a topic. The input is processed as a string to avoid any wrong data type manipulations and sent to the streaming module.



The screenshot shows the 'DASHBOARD' of the 'DISASTER TRACKER' application. It features a description of the tool's purpose: to generate insights from disaster-related tweets, increase situational awareness, and map tweets on a crowdmap. Below the description, there is a text input field for a disaster-related hashtag, with a sample input of '#AlaskaEarthquake'. A dropdown menu allows selecting the country where the disaster is happening, with 'United States' currently selected. A 'Stream Tweets' button is located at the bottom of the input section.

Fig 45. User input page showing user input fields and sample input

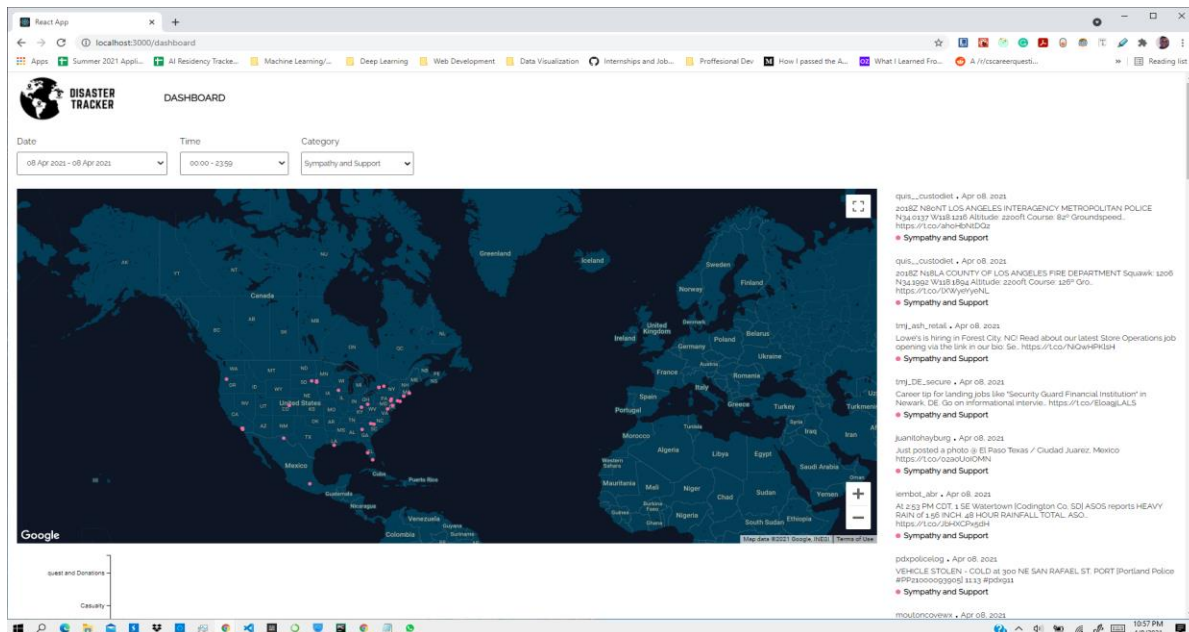
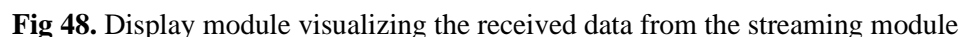


Fig 46. Dashboard showing data filters as an option list to restrict input

The streaming module is connected to the display module through API endpoints. The Disaster Tracker API provides 2 endpoints from which the display module can communicate with the streaming module to send data. The communication occurs in 2 ways, a POST and a GET request from the display module to the streaming module. To test the integration between the two modules, we input a hashtag and country in the display module input fields and submit. Submitting the input sends a POST request to the streaming module which sent a success message after its receipt. An automatic GET request is sent to the streaming module to collect any streamed tweets that have been stored. The streaming module responds with the requested data.

Fig 47. Response received by the display module from the streaming module



5.5 Analysis and Discussion

The evaluation results from the filtering model strongly suggest that by using machine learning models, we can accurately (with some degree of error) identify the relevance of tweets to disasters. This can be applied when collecting data from tweet streams e.g. the Twitter Streaming API, reducing the amount of noise present in the data and only allocating resources to relevant data. This will greatly improve the accuracy and speed of processing and analysing the data as focus is solely on the relevant data. For our filtering model, the model was biased towards irrelevant tweets. The model was much better at identifying irrelevant tweets compared to relevant ones. One possible reason for this could be the use of an imbalanced dataset. The training data had relatively more samples of irrelevant tweets compared to relevant ones and thus the model could have optimized its weights in favor of the irrelevant tweets as it had more samples to train on.

Results from the categorization model suggest that it is possible to use machine learning models to correctly and automatically categorize disaster-related tweets into humanitarian response activities based on their messages. This allows us to build a robust system that streams live disaster-tweets, filters out the noise and categorizes them accordingly. This process, despite its error rate, is much more time and resource efficient compared to using human resources to do these tasks.

5.6 Recommendations

Our recommendations for this system are to:

1. Humanitarian Organizations and Disaster Response Teams

Our system, Disaster Tracker, will allow humanitarian organizations and disaster response teams stay up-to-date with events happening during the early stages of a disaster. The system gives these teams an overview of prevailing events as they happen on a crowdmap tagged with the appropriate response activities that the messages should trigger from the team. The crowdmap allows the teams to quickly identify the locations of tweets to their exact street, roads and buildings. This is crucial as events during this stage are rapidly

evolving and teams need to get wind of the changes in real time. Gaining situational awareness for these teams is important as this allows them to better prepare and prioritize their resources based on the situation.

5.7 Future Work

In the future, these are the features that we intend to incorporate into the system to make it more efficient:

1. Tweet location extraction

Identifying the location of a tweet is difficult especially if location information is lacking in the tweet's metadata. New approaches to identify locations of tweets besides using the geotag information need to be implemented. These include identifying named entities in the tweets that correspond to locations or landmarks and getting their coordinates from a geocoding API to estimate the location of the tweets. Another way would be to estimate the location of a tweet from the user's previous locations. This will reduce overdependence on geotagged information to identify the location as very few tweets actually have that information in their metadata.

5.8 Conclusion

The aim of this project was to build a tool to help disaster response teams get better situational awareness of a disaster using tweets as an alternative data source. To do so, we built two machine learning models that process the tweets. The first identifies the relevance of a tweet just like a human annotator would do and discards irrelevant tweets i.e. tweets not related to a disaster. The second model categorizes the tweet as one of five humanitarian response activities. The models were deployed and integrated into a system that streams live tweets, processes them and maps the tweets on a crowdmap using their geographic location. Basic insights about the tweets and information they contain are analysed and displayed on a dashboard alongside the crowdmap.

Results from this project support the viability of Twitter and microblogs as alternative sources of information during a disaster for disaster response teams. The biggest challenge was the lack of geographical information on the majority of the tweets that were streamed hence a need for further research into how this can be improved. This project has paved the way for more work to be done to improve the efficiency of disaster response teams using Twitter as a source of data.

GitHub Repo

All the code from this project has been uploaded to GitHub. You can access the repository at the following link: <https://github.com/kelvinchumbe/Disaster-Tracker-Capstone-Project>

References

- [1] H. Seppänen, J. Mäkelä, P. Luukkala, and K. Virrantaus, “Developing shared situational awareness for emergency management,” vol. 55, pp. 1–9, 2013, doi: 10.1016/j.ssci.2012.12.009.
- [2] D. Johnson, L. K. Comfort, D. Johnson, A. Zagorecki, J. M. Gelman, and L. K. Comfort, “Improved Situational Awareness in Emergency Management through Automated Data Analysis and Modeling Improved Situational Awareness in Emergency Management through Automated Data Analysis and Modeling,” vol. 8, no. 1, 2011, doi: 10.2202/1547-7355.1873.
- [3] D. Yang *et al.*, “Providing real-time assistance in disaster relief by leveraging crowdsourcing power,” *Pers. Ubiquitous Comput.*, vol. 18, no. 8, pp. 2025–2034, 2014, doi: 10.1007/s00779-014-0758-3.
- [4] L. Cheng, J. Li, K. S. Candan, and H. Liu, “Tracking Disaster Footprints with Social Streaming Data,” *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 01, pp. 370–377, 2020, doi: 10.1609/aaai.v34i01.5372.
- [5] M. S. Dao, P. Quang Nhat Minh, A. Kasem, and M. S. Haja Nazmudeen, “A context-aware late-fusion approach for disaster image retrieval from social media,” *ICMR 2018 - Proc. 2018 ACM Int. Conf. Multimed. Retr.*, no. April, pp. 266–273, 2018, doi: 10.1145/3206025.3206047.
- [6] Y. Kryvasheyeu *et al.*, “Rapid assessment of disaster damage using social media activity,” *Sci. Adv.*, vol. 2, no. 3, 2016, doi: 10.1126/sciadv.1500779.
- [7] J. P. Singh, Y. K. Dwivedi, N. P. Rana, A. Kumar, and K. K. Kapoor, “Event classification and location prediction from tweets during disasters,” *Ann. Oper. Res.*, vol. 283, no. 1–2, pp. 737–757, 2019, doi: 10.1007/s10479-017-2522-3.
- [8] M. Imran, C. Castillo, J. Lucas, P. Meier, and S. Vieweg, “AIDR: Artificial intelligence for disaster response,” *WWW 2014 Companion - Proc. 23rd Int. Conf. World Wide Web*, no. February, pp. 159–162, 2014, doi: 10.1145/2567948.2577034.
- [9] A. Kumar and J. P. Singh, “Location reference identification from tweets during emergencies: A deep learning approach,” *Int. J. Disaster Risk Reduct.*, vol. 33, no. October, pp. 365–375, 2019, doi: 10.1016/j.ijdrr.2018.10.021.
- [10] A. Kumar, J. P. Singh, Y. K. Dwivedi, and N. P. Rana, “A deep multi-modal neural network for

- informative Twitter content classification during emergencies.pdf.” *Annals of Operations Research*, 2020, doi: <https://doi.org/10.1007/s10479-020-03514-x> S.I.
- [11] J. Rogstadius, M. Vukovic, C. A. Teixeira, V. Kostakos, E. Karapanos, and J. A. Laredo, “CrisisTracker: Crowdsourced social media curation for disaster awareness,” *IBM J. Res. Dev.*, vol. 57, no. 5, 2013, doi: 10.1147/JRD.2013.2260692.
 - [12] A. Karami, V. Shah, R. Vaezi, and A. Bansal, “Twitter speaks: A case of national disaster situational awareness,” *J. Inf. Sci.*, vol. 46, no. 3, pp. 313–324, 2020, doi: 10.1177/0165551519828620.
 - [13] S. Vieweg, A. L. Hughes, K. Starbird, and L. Palen, “Microblogging During Two Natural Hazards Events : What Twitter May Contribute to Situational Awareness,” *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2010.
 - [14] M. Kogan, L. Palen, and K. M. Anderson, “Think Local , Ret weet Global : Retweeting by the Geographically - Vulnerable during Hurricane Sandy,” 2012.
 - [15] H. Gao, G. Barbier, and R. Goolsby, “Harnessing the crowdsourcing power of social media for disaster relief,” *IEEE Intell. Syst.*, vol. 26, no. 3, pp. 10–14, 2011, doi: 10.1109/MIS.2011.52.
 - [16] A. D. Malawani, A. Nurmandi, E. P. Purnomo, and T. Rahman, “Social media in aid of post disaster management,” *Transform. Gov. People, Process Policy*, vol. 14, no. 2, pp. 237–260, 2020, doi: 10.1108/TG-09-2019-0088.
 - [17] K. Starbird and L. Palen, “Pass It on,” *Am. J. Nurs.*, vol. 115, no. 5, p. 7, 2015, doi: 10.1097/01.NAJ.0000465009.27201.47.
 - [18] C. Caragea *et al.*, “Classifying text messages for the haiti earthquake,” *8th Int. Conf. Inf. Syst. Cris. Response Manag. From Early-Warning Syst. to Prep. Training, ISCRAM 2011*, no. May 2014, 2011.
 - [19] A. Olteanu, S. Vieweg, and C. Castillo, “What to Expect When the Unexpected Happens : Social Media Communications Across Crises.”
 - [20] M. Imran, P. Mitra, and C. Castillo, “Twitter as a lifeline: Human-annotated Twitter corpora for NLP of crisis-related messages,” *Proc. 10th Int. Conf. Lang. Resour. Eval. Lr. 2016*, no. May, pp. 1638–1643, 2016.

- [21] M. Imran, C. Castillo, F. Diaz, and P. Meier, “Practical Extraction of Disaster-Relevant Information from Social Media,” pp. 1–4.
- [22] M. Imran, S. Elbassuoni, C. Castillo, F. Diaz, and P. Meier, “Extracting information nuggets from disaster- Related messages in social media,” *ISCRAM 2013 Conf. Proc. - 10th Int. Conf. Inf. Syst. Cris. Response Manag.*, no. May, pp. 791–801, 2013.
- [23] “Visualizing Seven Years Of Twitter’s Evolution: 2012-2018.”
<https://www.forbes.com/sites/kalevleetaru/2019/03/04/visualizing-seven-years-of-twitters-evolution-2012-2018/?sh=36e0ad287ccf> (accessed Dec. 09, 2020).