Data Fusion Tool for Environmental Hazard Detection in Switzerland

Master Thesis

Dana Rim Ghousson

University of Fribourg

December 2024

"Freedom of thought is the soul of progress."
– *May Ziadeh*

Acknowledgements

I would like to sincerely thank Prof. Dr. Jacques Pasquier for accepting my proposal to do the thesis in the Software Engineer Group and for the encouragement throughout the process. I am also very grateful to my supervisor, Dr. Mourad Khayati, for his feedback and advice, which helped me stay focused and improve my thesis. I want to thank my family and friends for their support and motivation and for always lifting my spirits. Finally, I would like to thank Dr. Käthi Liechti for providing a dataset from the Swiss flood and landslide damage database.

Abstract

Disaster detection is a significant challenge around the globe. In Switzerland, floods, debris flows, and other natural hazards pose a risk to citizens and infrastructures. With the increasing availability of multimodal social media data, there is a potential to enrich numerical data from sensors and weather stations. Existing disaster detection solutions rely mostly on a single source of data, which limits their applicability.

This thesis introduces a mobile application for disaster detection. It implements a data fusion algorithm that combines time series, textual, and geo-location data extracted from Twitter. The application allows to seamlessly to detect new natural hazards and insert hazard warnings manually, promoting community engagement and faster responsiveness. The evaluation of the system is achieved through accuracy tests with a controlled dataset. The resulting accuracy is around 80 percent for floods and earthquakes under a specific configuration of the algorithms. The code parts of the mobile application are further tested using unit tests.

Keywords: Hazard detection, data fusion, multimodal data, mobile application

Dr. Mourad Khayati, Software Engineering Group, Department of Informatics, University of Fribourg (Switzerland), Supervisor

Prof. Dr. Jacques Pasquier-Rocha, Software Engineering Group, Department of Informatics, University of Fribourg (Switzerland), Co-supervisor

Table of Contents

List of Figures

List of Tables

Code Listings

List of Algorithms

1. Introduction

1.1. Motivation and goals

Today, social media has become part of everyday life for a significant portion of the world's population. Twitter, Instagram, Facebook, and many other social media platforms not only significantly impact how people communicate and publish experiences but have also become major collectors of huge amounts of diverse data. This data includes various modalities like text, video, images, and geolocation. The accessibility of such platforms presents an ideal opportunity to use the data for objectives beyond social networking - including optimization of natural disaster management and alert systems.

Human lives and infrastructures can suffer bad impacts from natural hazards such as floods, fires, and earthquakes. Hence, disaster detection is very important and relies on prompt information. Traditionally, numerical data from weather stations and sensors are collected and used for detection systems. While this data is invaluable, it often lacks context of the situation that social media data can give. Posts from social media can give real-time updates on developing natural hazards and capture observations from individuals on the ground. There can be descriptions of disasters in text format, images, videos visualizing the event, or geolocation information on where the disaster occurs. Focusing on this multimodal data could optimize the effectiveness of disaster detection and alert systems.

The main goal of this master thesis is to collect historical numerical data on events of natural hazards that happened in Switzerland and to combine them with social media data, precisely tweets about natural hazard events. Switzerland is affected by floods, avalanches, storms, earthquakes, and other natural hazards, and this is an exemplary context for studying such an application. By fusing real-time social media data with historical numerical data, a model can be implemented to optimize disaster detection. The detection algorithm uses the fused data to reveal new hazard events and alert people when new hazards arise. This approach has the potential to allow dynamic and proactive responses to disasters.

To ensure accessibility and user-friendliness for this system, a mobile application is implemented as part of this master thesis. The application visualizes the fused data so that users can check on alerts in their regions and display descriptive data about new hazards. The benefit of having social media data included is that users have access to tweets describing the situation. This can help people understand the severity of the event and empower them to decide faster about their safety. In addition, the application contains a feature allowing users to manually insert alerts instead of using social media networks. This should ensure that users can send warnings immediately when a hazard happens. It allows a certain engagement of the community and guarantees a robust system in moments where social media data might be sporadic or delayed.

1.2. Research questions

The following questions were formulated for this master thesis:

- How can social media data enrich numerical natural hazard data?
- How can social media be used to provide effective alerts during natural disasters or environmental hazards?
- What are the key elements that encourage user interaction and engagement during natural hazards?
- What are the fundamental components and best practices for designing a software architecture for a hazard detection system?

1.3. Outline

Chapter 2 shows related works of this thesis. On the one hand, papers are presented using similar technologies and methods for detection using social media data, and on the other hand, applications are presented that are currently in use in Switzerland. Chapter 3 explains the different datasets: the historical data, the warning data, the social media data, and the controlled data. In the social media data part, a Twitter scraper is described. Additionally, it presents the data fusion theory and one algorithm, the SVD-based Multimodal Clustering Method for Social Event Detection, which is used in the application developed for this thesis. Chapter 4 introduces the whole implementation of front- and backend. The backend is composed of a Python application that does data fusion and detection calculations and Twitter scraping. A PostgreSQL database stores all the past and new natural hazard events. On the other hand, the frontend application creates the bridge from the application to the user. A mobile application for Android is presented where users can be alerted of new natural hazards and where users can report new natural hazards in the application. Finally, Chapter 5 summarizes the research questions and examines if they were answered and if the goal was reached, and an outlook is presented.

1.4. Technologies used

The backend is created with Python and the micro-framework Flask. It uses a PostgreSQL [30] database and a Selenium [33] scraper. To run it, a Docker container is created. The frontend is written in Dart with the Framework Flutter [17]. An APK is created to run the frontend.

2. Related work

In this chapter, we review key research papers in the data fusion field and present existing natural hazard alert applications.

2.1. Research papers

Many papers in recent years have explained how to detect new events using techniques such as natural language processing, machine learning, and social network analysis.

Okazaki and Matsuo [6] presented a new approach to using social media data for real-time event detection of earthquakes. The authors introduced the term "social sensors", to describe social media users collecting data about new events. During earthquakes, social media users often report their experiences on platforms and thus create data that can be used to detect new earthquakes. The author's system collects specifically geotagged Twitter data related to earthquake-related keywords. The data is then analyzed with a probabilistic model to decide if a tweet determines a real earthquake or just a metaphorical phrase about earthquakes. Their study showed that, in some cases, their system could identify earthquakes faster than traditional detection methods, which gives social media data additional importance in early detection methods. Their paper inspired research in detection methods using social media data and the potential of people-generated data.

Another system was introduced in [7] to detect all kinds of events using Twitter streams. The proposed system analyses words by applying an EDCoW (Event Detection with Clustering of Wavelet-based Signals). The algorithm creates signals for each word which are then calculated by wavelet analysis. Trivial words are filtered and cross-correlations between the signals are measured by the EDCoW. These signals are then clustered into events using modularity-based graph partitioning. Their experiments show promising results of this algorithm and confirm the importance of social media data in event detection.

TEDAS [3] is a Twitter-based event detection and analysis system that retrieves social media data through the Twitter API. It collects them with a query having a spatial range, time period and keyword about "crime and disaster-related Events (CDE)". The tweets are then clustered into events based on location and time patterns and visualized on a map.

Another Twitter-based event detection analysis system was developed by Xia et al [8]. Its goal is to show CityBeat users information about real-time events and "alert them to unusual activities". Geotagged social media data is collected and going through "time series analysis and classification techniques". The time series analysis is implemented through a predictive model where Gaussian Process Regression is used. Additionally, an alarm engine is implemented with the assumption that more photos would signify an event. If the number of photos of a certain region is much higher than normal it is labeled

as a "candidate event". These candidate events will then go through a classification algorithm, here a binary SVM with 22 features, and be labeled as event or non-event. Similar to Xia et. al, Geoburst [9], a real-time local event detection that uses geo-tagged tweets, collects data from Twitter and clusters them depending on their location and semantic similarities. These clusters are seen as candidates and go through a ranking system where local events are ranked higher and global topics lower.

More recently, a social fusion algorithm was presented [2]. The system uses crawlers to get Twitter and Instagram data daily by API calls. A tweet clustering module is applied to remove redundant tweets and an Instagram event localizer to cluster Instagram posts. Both data undergo an EM-Fusion, fusing both types of data and generating output events. This approach highlights the benefits of using two kinds of social media data: text from tweets and pictures from Instagram.

2.2. Applications

There exists already applications using numerical data to alert people about natural hazards. Here, two applications used in Switzerland are presented.

The MeteoSwiss Natural Hazard Map [24, 23] is a warning map offered by the Federal Office for Meteorology and Climatology MeteoSwiss. It shows warnings for Switzerland and Liechtenstein. Several types of warnings are grouped by severe weather or natural hazards. Severe weather includes "wind, thunderstorms, rain, snow, hard-packed snow, heat and frost" and natural hazards include "floods, forest fires, avalanches and earthquakes". It is also possible to see warnings about wind for lakes and airports. The map uses colors to show the danger level, which ranges from 1 to 5, with 5 being the greatest danger. The region will be marked with hatches for events with a very low probability.

The warning data is prepared by various specialized agencies like the MeteoSwiss, The Federal Office for the Environment, the WSL Institute for Snow and Avalanches Research, and the Swiss Seismological Service. To predict new hazards, the agencies use "high-resolution numerical forecasting models". It is possible to display the map on their website or the mobile app "MeteoSwiss app" [25]. On the mobile app there is a feature to subscribe to certain regions and be notified only for these.

Another application employed by many people in Switzerland is the AlertSwiss [10] mobile app, "an alert and information channel operated by the federal government and the cantons". Warnings are sent by agencies in charge of incident management. The notifications are categorized into three groups: alert, warning and information. Users receive them as push notifications and can see the warning in the application on a map and as a list. It is possible to choose specific cantons, to only be notified for these. [11]

The review of existing research papers highlights the progress made in social media data fusion, and the applications show the importance of alert applications, especially in Switzerland. The existing applications use numerical data and therefore lack multimodal data. Additionally, no user interaction is available. With this knowledge, a new mobile application is implemented in this thesis. To enhance the alerts, it is a focus of this work to add other modalities such as location, date, and text from social media posts to the numerical data. Similar to the explained research papers, the goal is to retrieve data from social media platforms such as Twitter, use a data fusion algorithm, and cluster the

data points to detect new natural hazards. The scraped tweets will serve as "candidate events" [8] and "social sensors" [6], as described in the respective papers. The fusion enables clustering of the data to detect if a candidate is a true event. A second feature will be implemented to allow user interaction with the application and to insert new alerts directly through the application.

3. Data Collection and Fusion

3.1. Data Collection

The first part of this thesis focuses on collecting data from past natural hazards in Switzerland. To narrow the scope of the thesis, we restrict the collection to Switzerland. The country is especially affected by natural hazards, such as floods and debris flows, landslides, fall processes, avalanches, and storms. Moreover, it experiences "forest fires, drought, heat and cold waves". While strong earthquakes rarely occur, they can still pose a significant potential risk [12]. Thus, the focus was set on hazards like floods, earthquakes, forest fires, and avalanches. In addition, it was important to collect data about natural hazards in Switzerland from social media to enrich the numerical data with other modalities.

3.1.1. Historical data

The historical data describes data from past natural hazards. Two datasets were collected, which are explained in the next sections.

WSL. The Swiss Federal Institute for Forest, Snow, and Landscape Research (WSL) focuses on environmental dynamics and landscape protection. Since 1972, more than 27'000 reports have been stored in its "Swiss flood and landslide damage database" [28]. The data supports hazard assessment, and on an interactive map, the damages that happened over time are illustrated.

The structure of the data is a CSV of 35 columns with information about the hazards' location, date and time, type, severity of damages, duration, rain quantity, and number of casualties. It consists of data from 1972 until 2023 from Switzerland and has three main hazard types, namely fall, landslide, and water/debris flow. There may be multiple entries for one hazard which are marked with the same number in the column "Grossereignisnummer, mehrere Ereignisse, welche aufgrund meteorologischer oder räumlicher Gegebenheiten zusammengefasst werden." The most important pieces of information taken from this dataset are the "x-Koordinate" and "y-Koordinate" (in EPSG:21781), "Datum", "Zeit" and "Gemeinde" columns. To make the data more structured, the data was split into multiple CSV lists for each hazard type, and all duplicate entries were removed.

SED ETH Zurich. The Swiss Seismological Service (SED) is the federal agency for earthquakes and is run by the ETH Zurich[31]. The earthquake dataset [32] was downloaded from their website. The dataset contains more than 4800 entries with information about time, magnitude, location, depth, latitude and longitude, assessment, and the reporting agency. The years vary from 250 to 2023. It contains earthquakes from Switzerland and countries around it, such as France, Italy, Liechtenstein, Germany, and Austria. The magnitude and size of an earthquake range from 2.5 to 6.6.

Earthquakes are assigned danger levels. According to the SED level 1 means little to no danger and has a magnitude of approximately 2.5 or greater. Earthquakes with magnitudes of around 3.5 or greater are assigned danger level 2, meaning moderate danger, and with magnitudes of approximately 4 or higher the danger level rises to 3 (significant danger). An earthquake is only classified into high danger with a magnitude of approximately 4.7 or greater and the highest danger level 5 will be assigned to the ones with a magnitude of 5.4 or greater. A level 3 can already damage buildings; there might be falling objects, and people will be alarmed. Level 4 can collapse structures with smaller stability, and people might lose their balance. Level 5 can destroy stable structures, and people lose their balance.

3.1.2. Social media data

Social media data is extremely beneficial when complementing numerical data. Numerical data delivers precise data but often lacks descriptive details in cases where people need to comprehend and react rapidly. When fusing descriptive social media data like text data with numerical data, people's awareness can be increased, particularly during a natural hazard [5].

To get data from past hazard events in Switzerland, Twitter was tested using a Twitter scraper. The reason for using a scraper instead of the Twitter API was due to the discontinuation of free API access for students and researchers. Employing a scraper supplied a cost-effective, open-source alternative for data collection.

The scraper utilized is Godkingjay's Selenium Twitter Scraper from GitHub. It can scrape tweets from home, user profile, hashtag, query or search, and advanced searches [34]. To use the scraper, one must have a Twitter profile and insert the username and password to the credentials.txt, as Twitter only allows logged-in users to display the tweets. One downside of using scraping is that Twitter can recognize activity and change the steps to log in, therefore stopping the program from entering Twitter. Only after manually solving the problem can the scraper continue.

The scraper was tested with queries about past natural hazards from the WSL dataset. The queries listed below show a location name or a river and a timespan when the hazard happened. At this moment, it has not yet been discovered that Twitter also allows to query for specific locations. With these queries, the query text must be mentioned in the tweet itself to make a match. For example, the tweet must contain the word "fribourg" if using the query "fribourg until:2023-11-15 since:2023-11-13". By specifying the location, it would also have been possible to find all the tweets about the event, even if they did not mention the word. Fortunately, there were still tweets mentioning these words and describing the past natural hazard event. After inspecting the tweets scraped with these queries, Twitter seemed a good choice for fetching data from past events and presumably also obtaining tweets about new hazards.

- arve until:2023-11-15 since:2023-11-13
- fribourg until:2023-11-15 since:2023-11-13
- gürbe until:2023-11-15 since:2023-11-13
- rhein until:2023-11-15 since:2023-11-13
- arve until:2023-12-12 since:2023-12-11

- sisikon until:2019-07-29 since:2019-07-27
- lenk until:2018-07-28 since:2018-07-27

The Twitter scraper was thus adapted to the needs of the project. To identify the location of a tweet, the user must allow it to be in the Twitter settings. Because only 1.5% of the tweets are geotagged, it was chosen to also include a second approach for finding the location [6]. As documented by Twitter, 30-40% of Twitter users also mention their location in the profile [38]. Therefore it was decided to also scrape the profile location. The original Twitter scraper did not include both options, so they were implemented. As visualized in Code Listing 3.1 the first part is the scraping of the location via a Tweet. The XPath searches in the HMTL for an element <a>, which defines a link that contains '/places/'. It will then output the text inside the element. The second part is the scraping of the location from the user's profile. It searches for an element with parameter @data-testid='UserProfileHeader_Items' and dives into three spans. The last span contains a text, which will be the resulting location from the profile.

To get the location from the tweet, the scraper must redirect the URL inside the tweet to see the location. To get the location from the profile, the scraper needs to redirect to the profile page. Between changing the routes, sleep functions have been added because otherwise, the scraper might be too fast and scrape zero tweets if the page did not finish to load yet. It also happened that the URL of tweets did not work, so it was not possible to get the location information.

```
try:
    self.twitter_loc = self.card.find_element(
        "xpath", ".//a[contains(@href, '/places/')]//span"
    ).text
except NoSuchElementException:
    self.twitter_loc = "noLocInTweet"

try:
    self.twitter_loc_profile = self.card.find_element(
    "xpath", "//div[@data-testid='UserProfileHeader_Items']//span//span//span"
    ).text
except NoSuchElementException:
    self.twitter_loc_profile = "noLocInProfile"
```

Code Listing 3.1: Scrape Twitter location information

After discovering the ability to use the "geocode" parameter in Twitter queries, it was included in the query. Unlike including the location name as plain text, the "geocode" parameter allows for defining a geographic area by adding coordinates and a radius. This makes it possible to fetch tweets exclusively from a fixed circular region. As visualized in Code Listing 3.2, the query is composed of a keyword for natural hazards (represented as the variable "query"), the geocode, and a date parameter specifying tweets posted since yesterday. The latitude, longitude, and radius included in the query encompass the entire area of Switzerland and were calculated with the tool Map Developers [22].

```
 # Get today's date
today = datetime.today()

# Get yesterday's date
yesterday = today - timedelta(days=1)

# Format date as 'YYYY-MM-DD'
yesterday_str = yesterday.strftime('%Y-%m-%d')

query = query +" geocode:46.726266,8.124314,182km since:"+yesterday_str
```

Code Listing 3.2: Query with geocode and date

3.1.3. Synthetic Datasets

Two controlled datasets, one based on the WSL dataset for floods and one based on the SED earthquake dataset, were created to test the application with natural hazard data instead of the data proposed by the paper [4]. The mentioned paper will be presented in Chapter 3.2.1.

Both datasets were generated using a small Python script. The formerly controlled dataset generates for each entry in the WSL dataset randomly 2 to 4 tweets. It combines the time and date information into one field. If no time was indicated, it will be added "00:00:00" to the entry. So that the tweets have different times, we add random hours from 0 to 11. When reaching "23:59:59," it will go back to "00:00:00," and it counts further from there. The date stays as it is. The field first shows the date and then the time, for example, "2024-12-12 12:10:00".

The X- and Y-coordinates are transformed into latitude and longitude. In the original dataset, the coordinate system is EPSG:21781 and will be changed to EPSG:4326. Latitude and longitude will each be saved as one field. To create a text for the tweet, parts of sentences were randomly chosen from the floodList in Code Listing 3.3, and the municipality was added at the end. The list contains text about the flood in four languages: German, French, Italian, and English. For example, "Hochwasser in Bern". Note that the municipalities can be in different languages. For example, "Basel" is written in German, but "Genève" is in French. To make it similar to the dataset from the algorithm, a username was added to the tweets. This information was later chosen to be ignored in the implementation because it would only give imprecise information.

```
floodList = ["{\"U}berschwemmung in ", "Hochwasser in ", "indondation {\`a} ", "crue
    {\`a} ", "inondazione a ", "alluvione a ", "flooding in ", "high water in "]
```

Code Listing 3.3: List with flood text

Similar steps were taken for the second controlled dataset about earthquakes. For missing date and time, the empty field was filled with the first month "January", the first day of the month "1" and "00:00:00" for the time. Only earthquake data with a magnitude of at least 3.5 was fetched. This means earthquakes with a danger level of 2. This was chosen because, from this level of danger, people are supposed to take cover and be prepared for further earthquakes. Those data points without location were left out. The locations were already stored as latitude and longitude; therefore, no transformation

had to be done. Instead of using a list of flood words, the list below in Code Listing 3.4 was used:

```
earthquakeList = ["Erdbeben in ", "Tremblement de terre Ã  ", "Terremoto a ", "
    Earthquake in "]
```

Code Listing 3.4: List with earthquake text

It was decided that the text of the tweets in the two datasets should contain the type of the event and the location. For example, a text would be "Erdbeben in Bern". Where "Erdbeben" is the type and "Bern" is the location. This decision was made after analyzing real tweets from the past recorded events found in the WSL dataset. It was observed that people often reference the location directly or use names such as river or lake names to describe the event.

3.2. Data Fusion Theory

Social media data is often multimodal, meaning that there are different modalities like text, image, video, and geolocations. The theory of data fusion is to combine information from different modalities and sources together to ensure better performance and to enrich datasets with additional knowledge [1]. In Figure 3.1, an example of such a data fusion is shown. Notes, time series, imaging, omics, and structured data are fused into a single model to then do decision-making or predictions with regression, classification, or clustering [26]. The objective is thus to use five different modalities to improve decision-making and predictions.

Data fusion can be performed at three different stages: early, intermediate, or late. Early fusions combine modalities without preprocessing them. The fused data can then be inserted into the model. Intermediate fusion, on the other hand, needs each modality to be preprocessed into a "latent representation" before being used for the model. Last, the late fusion method processes each modality on its own and is independently inserted into a model. The output of the different models will then be fused at a later moment [27].

3.2.1. SVD-based Multimodal Clustering Method for Social Event Detection

The selected data fusion algorithm is an SVD-based Multimodal Clustering Method [4]. It uses four different modalities, and the data fusion is performed at an early stage. It was developed to detect social events in an unsupervised approach, with multimodal K-means clustering using SVD. The results of the method were tested with the MediaEval SED 2012 dataset, and good results were obtained. This dataset consists of Flickr images accompanying metadata, and the paper's goal was to cluster images from the same events together. The chosen modalities for the algorithm are timestamps, tags, geo-tags, and usernames. With timestamps, tags, and geo-tags, good results were achieved, and even with location absent, 80 percent of good results were obtained.

As shown in Algorithm 1, the technique is structured in six steps and needs N data points, the number of nearest neighbors for each modality, the reduced dimension D, and the number of clusters C to start. At step 1 the adjacency matrix is calculated for each

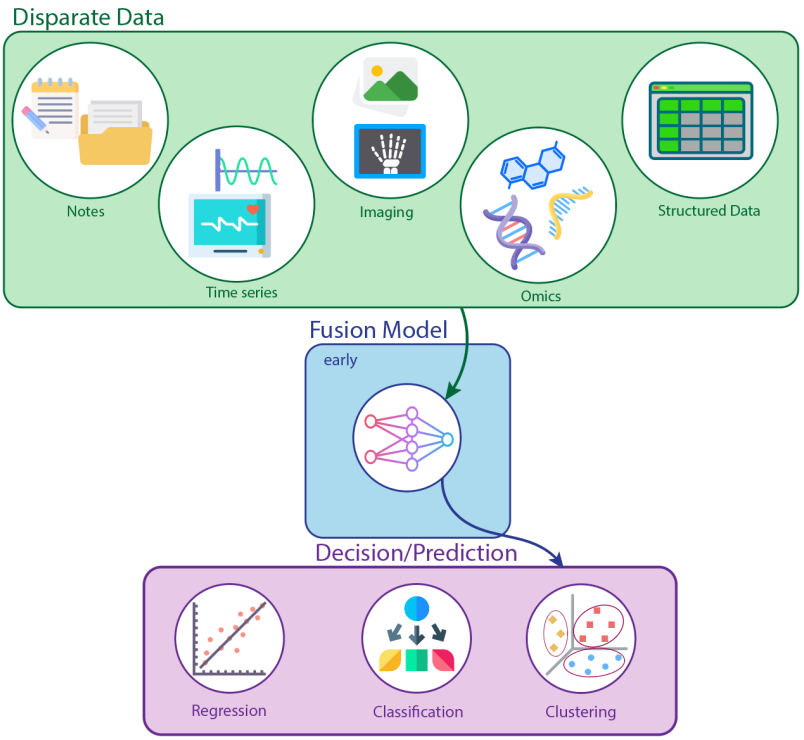


Figure 3.1.: Multimodal data fusion [26]

modality. This means every data point is compared to the other $N-1$ data points and results in an adjacency list of the nearest neighbors marked as 1 and otherwise 0. The step was slightly changed to also add a 1 to itself. Like this, the lists are more similar when a data point x has y as a neighbor and vice versa. The calculation is done for each modality and each data point. A specific technique was used for each modality to calculate the differences between the data points. They will later be explained in detail in Table 4.3. Step 2 merges the lists of each modality to get one list per data point, resulting in a multimodal adjacency matrix. Step 3 calculates the SVD for the adjacency matrix using the reduced dimension D. With $U'$and the $\sum'$, the feature matrix can then be calculated in step 4. The feature matrix is finally inserted in a K-Means Algorithm with C clusters and provides a list of labels l, where each data point is part of a cluster with a label.

To showcase the algorithm in an example, two data points belonging to one event are shown on the next page. The two data points only differ in one tag (Linux and Windows) and the usernames (BillGates and AnonymousPerson).

Algorithm 1 The SVDMC Algorithm.

Input:
A multimedia dataset with N objects;
The numbers of nearest neighbors for each modality $\{k_m\}$;
Objective reduced dimension D;
The number of clusters C.
Output:
Cluster indicator vector l.

1: For each modality m, compute the adjacency matrix $A^m \in R^{N \times N}$, where $A^m_{ij} = 1$ if and only if x_j is among the k_m nearest neighbors of x_i .
2: Construct the multimodal adjacency matrix A by operating logical OR on the unimodal adjacency matrices.
3: Conduct SVD on the fused adjacency matrix to find a low-rank approximation of A, i.e. $A' = U' \sum' V^{T'}$, with the largest D singular values in $\sum'$ and corresponding singular vectors in U' and $V^{T'}$.
4: Generate the new feature matrix $F = U' \sum'$.
5: Cluster the rows of F into C clusters using K-means and get the cluster indicator vector l.
6: **return** l;

```
data = [
              {"@id": 0,
              "tags": {"tag":["tech", "event", "Linux"]},
              "location": {
                        "@latitude": 46.8676734812958,
                        "@longitude": 7.81247335867426
                        },
              "@dateTaken": 2024-01-28 12:00:00,
              "@username": BillGates
              },
              {
              "@id": 1,
              "tags": {"tag":["tech", "event", "Windows"]},
              "location": {
                        "@latitude": 46.8676734812958,
                        "@longitude": 7.81247335867426
                        },
              "@dateTaken": 2024-01-28 12:00:00,
              "@username": AnonymousPerson
              },
      ]
```

The algorithm begins by calculating the nearest neighbors for each data point and modality. We configure the modalities to calculate one nearest neighbor per modality. If there is no username similar to the one from the data point, the list of nearest neighbors will stay empty. Since we only have two data points, the nearest neighbors for the three modalities will be the other data point.

```
[
    {"location":[1],"tags":[1],"@dateTaken":[1], "@username": []}, #id=0
    {"location":[0],"tags":[0],"@dateTaken":[0],"@username": []}, #id=1
]
```

The next step is to create the adjacency matrices for each data point and each modality. Below, each row represents a modality, and each column is a data point. For example, the row [[1,0], [0,1]] shows that in the first list [1,0], only the data point 0 is the nearest neighbor. This is because it is compared to itself. The second data point instead is zero, meaning data point 0 and data point 1 do not share a username.

```
[
    [[1, 1], [1, 1]], #@dateTaken
    [[1, 1], [1, 1]], #location
    [[1, 1], [1, 1]], #tags
    [[1, 0], [0, 1]] #@username
]
```

To combine the modalities, the lists are unified into one, which gives us the resulting adjacency matrix. Both lists are identical, showing the similarity between the two data points.

```
[
    [1, 1], #id=0
    [1, 1] #id=1
]
```

If there were more data points, the adjacency matrix would be bigger and more informative. This matrix would then undergo an SVD and the dimensionality would be reduced. The feature matrix resulting from it can then be inserted into a clustering algorithm, for example, the k-means. For this method, the number of clusters must be defined beforehand. The output of the clustering would then give each data point a label.

For example, if the matrix below is undergoing the process from SVD to the clustering, with the number of clusters of two, the result would be that data points 0 and 1 will be in one cluster and data points 2 and 3. This is because of the similarity of the lists.

```
[
    [1, 1, 0, 0], #id=0
    [1, 1, 0, 0], #id=1
    [0, 0, 1, 1], #id=2
    [0, 0, 1, 1], #id=3
]
```

4. Mobile Application

4.1. Architecture

The primary goal of implementing this mobile application is to ensure that users are alerted when newly occurring natural hazards happen and to enable them to report such hazards. As depicted in Figure 4.1, historical hazard events stored in the database undergo a data fusion procedure. Concurrently, new hazards will be filtered from Twitter with specific keywords and will similarly undergo a data fusion. After the data fusion, the new and historical data will be merged and analyzed using Singular Value Decomposition SVD and a clustering algorithm Density-Based Spatial Clustering of Applications with Noise DBSCAN to determine whether a new data point is a natural hazard. The new data points are accordingly labelled and saved in the database. The mobile application displays all detected hazards. It uses a routing mechanism and HTTP requests. Users will be notified in real-time when the new data becomes available in the mobile application. In addition, users are also allowed to report natural hazards through the mobile application, thereby contributing to prompter alerts and improved reaction time for other users and authorities.

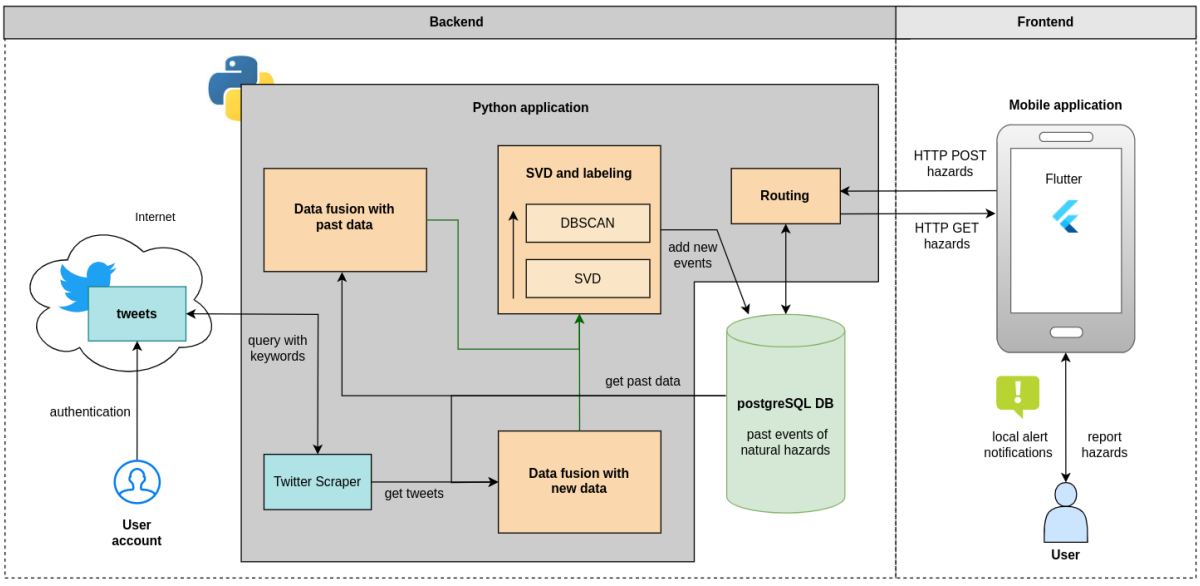


Figure 4.1.: Mobile Application Architecture

4.2. Backend Implementation

The backend of the mobile application is implemented in the programming language Python and employs the framework Flask for essential features such as routing and re-

quest handling. The algorithm of data fusion, as explained in the earlier chapters, and the detection method are developed alongside the mechanism of Twitter scraping. The connection between the backend and the frontend is established through Flask's routing mechanism.

4.2.1. Configuration

To execute the backend application, two parameters must be specified. By default, the run state is set to test mode (instead of real mode), and the amount of data retrieved from the database to be calculated inside the algorithm is set to 500 for demonstration reasons. As a performance reference, calculating 500 data points in the data fusion algorithm takes 2.84 seconds for flood and 2.59 seconds for earthquake data. For 1000 data points, the calculation time is increased to 11.27 seconds for flood data and 12.55 seconds for earthquake data. With 3000 data points, the calculation takes 120.59 seconds for flood data and 136.39 seconds for earthquake data. The computations were completed on a machine supplied with 16 GB of RAM.

4.2.2. Docker and APK

Docker containers [15] are created to simplify the setup of the backend application with the database. The advantages of a container are that it has one bundle of all dependencies and does not produce problems with different local environments. The directory structure of the dockerized folders is illustrated below.

```
project
├── backend
│   ├── Dockerfile
│   ├── requirements.txt
│   └── ...
├── db
│   ├── db_dump.sql
│   └── Dockerfile
├── docker-compose.yml
└── ...
```

The containers are defined in the docker-compose.yml file. To ensure the program runs on different modes, docker profiles are configured: a test and a real profile. The test mode will execute the backend application with a test data input, and the real mode will run the backend with the real Twitter data. The default number of the data being calculated from the database is set to 500 for demonstration reasons. The backend runs on port 8000 and the database is on port 5432. These must be consistent in the docker-compose.yml, the code and the Dockerfiles.

The database Dockerfile copies the db_dump.sql into the PostgreSQL database and exposes it on port 5432. The dumb contains all the controlled WSL flood data and controlled earthquake data. The Postgres user, password, and database name must be defined.

As the scraper is included in the backend application, the backend Dockerfile installs Firefox as a web driver and copies and installs everything from the requirements.txt file

into the container. It must be specified that the backend is exposed on port 8000. The reason for including the scraper in the same container as the main backend application is to avoid issues when attempting to call the scraper from the backend application. The problems arose due to invocation from different containers.

On the other hand, the frontend application is packed into an APK (Android Package Kit) file to ensure execution without installing dependencies. It is created with the command:

```
flutter build apk
```

Code Listing 4.1: Flutter APK command

The reason for choosing an APK instead of a Docker container was to ensure the frontend application ran on virtual devices.

4.2.3. Routing

Flask [16] defines the routings connecting the frontend application with the backend application. There are four routes specified in the program; notice Table 4.1.

id	HTTP method	route	parameters	function
1	GET	/api/hazardsPerYear	type of hazard	Returns the numbers of a specific type of hazard per year.
2	GET	/api/alertData	type of hazard	Returns all hazards from the last 24 hours.
3	GET	/api/tweetData	-	Returns all tweets from the last 24 hours.
4	POST	/api/data	JSON of new hazard	Saves a new hazard to the database.

Table 4.1.: Routing methods

The first route is employed to calculate the number of natural hazards that occur every year. The type of natural hazard can be specified. The route returns a JSON with an array of the years and the number of hazards for each year.

The second route returns all the hazard data of a specific type. It is named alert data because, depending on this data, notifications will be created for the mobile application and notify users. Only the hazards from the last 24 hours are retrieved.

The third route returns all the tweets from the last 24 hours that are connected to a natural hazard in the database.

The fourth route is used to add new hazards to the database. This is especially needed when users report new hazards via the mobile application. These hazards are labelled true for the "accepted" column in the database as default. This means that when users report hazards, they will automatically be seen as true events and not undergo an algorithm.

This presents a vulnerability when users misuse the submission. It must be addressed in the future.

4.2.4. Data Fusion and Detection

The base of the data fusion and detection is the algorithm presented in Chapter 3.2.1. The algorithm was first implemented with the data from the paper and then adapted to the natural hazard data.

The application runs the data fusion and detection twice. The reason is that each type of hazard will run on its own data and new tweets. The chosen two types are flood and earthquake. Initially, the two types will start calculating their past data. The data is stored in the database and obtained through a query call with SQLAlchemy, as seen in Code Listing 4.2.

```
results = db.session.query(HazardReport, Tweet)
                    .join(Tweet, HazardReport.id == Tweet.hazard_report_id)
                    .filter(HazardReport.type == type.capitalize()).all()
```

Code Listing 4.2: SQLAlchemy query historical data

The data fusion algorithm considers three modalities: the location, the tags, and the datetime. The modality username that was used in the paper was excluded. The reason is that news accounts were found in the data analysis, posting about different hazard events without them being connected. So this would only drift data points from the same events and locations apart rather than connect them. For each modality, a number of neighbors are chosen. The chosen numbers for a dataset of 500 data points are seen in Code Listing 4.3. The reduced dimension is set to 100.

```
k_m = {"location": 10, "tags": 3, "@dateTaken":1}
```

Code Listing 4.3: modalities iff N = 500

The number of neighbors chosen was analyzed through an accuracy function and comparison of maps created with folium. [19] Note that the test data is part of the data. Of the 500 data points, 100 were chosen, which resulted in 20 percent of test data. As it is the same data, we must consider that the dates will not represent new data, which will have more recent dates. As shown in Table 4.2, the accuracy test was made with the consistent DBSCAN parameter epsilon = 2. The reason for using this clustering algorithm with this epsilon parameter is explained later.

For the three modalities, different values were chosen and compared. The best results are achieved with only the location or only the date modality. This can be attributed to the fact that the tweets close to each other belong to the same events, as well as those that were created around the same date. As new tweets will not have the same date as past events, less emphasis will be placed on the date modality. Therefore, the best accuracy would be achieved by only weighing the modality location. Because there might be tweets sent from other locations about the same events, the modality tags are also valuable. For example, someone tweeting from Zurich about a flood in Zermatt. Hence, the modality tags were chosen to have a weight of 3. With location = 10, tags = 3 and date = 1 we achieve a result of 0.88 resp. 0.84 accuracy with the DBSCAN epsilon set to

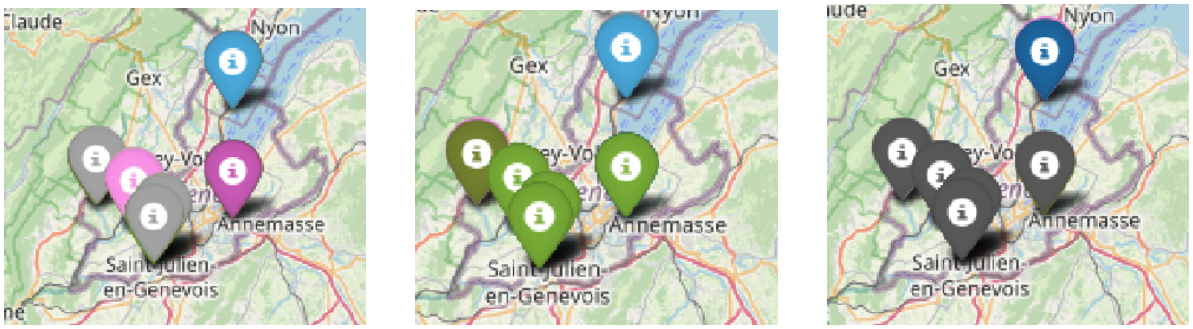
2. The modality location gets the highest importance in clustering past events from the same locations together. It was assumed that hazards happen in the same spots more often than in new spots. For example, flooding would happen more next to rivers and lakes than somewhere with no water.

N	test N	loc	tags	date	epsilon	ac test flood	ac test earthquake
500	100	10	10	10	2	0.12	0.09
500	100	10	10	null	2	0.46	0.24
500	100	null	10	10	2	0.05	0.07
500	100	null	10	null	2	0.52	0.79
500	100	null	null	10	2	0.9	0.98
500	100	10	null	null	2	0.92	0.88
500	100	10	3	null	2	0.88	0.85
500	100	10	3	1	2	0.88	0.84

Table 4.2.: Modality weight accuracy test

As illustrated in Figures 4.2a to 4.3c, two locations were compared for different epsilons in the DBSCAN. The DBSCAN is the clustering algorithm, which will be presented in detail later and is used to label the data points. The first example is Geneva. An epsilon of 1.1 creates five different clusters. For epsilon 1.5 instead, it creates 3, and for epsilon 2, only 2 clusters. The data point near Saint-Julien-en-Genevois and the one near Annemasse have a distance of almost 20km. This is quite far for people, so we suggest clustering these data points into different events. This is in the case of epsilon 1.1.

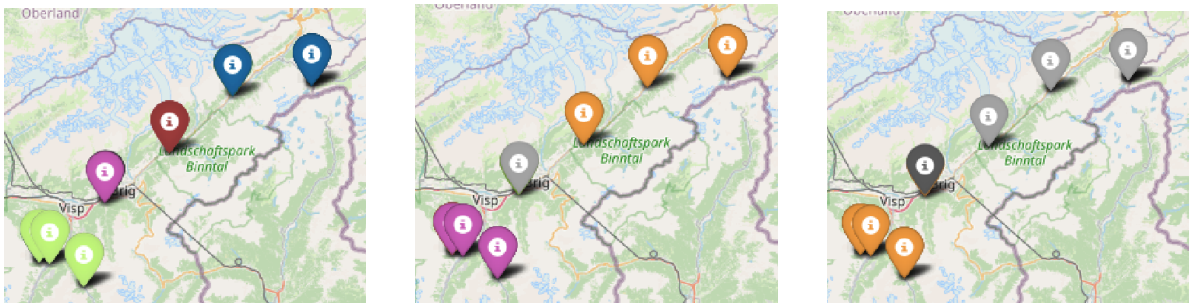
The next example is in Visp. Epsilon 1.1 clusters the data points into five labels. Epsilon 1.5 and 2 are both into 3 clusters. The distance from Visp to the last data point on the right is almost 50 km away. Therefore, in Figure 4.3b, the rightest orange data point and the most left orange have circa 25 km distance. In this case, it would be better to take epsilon 1.1 so that the distance of the data points is less and the red data point stays as its own cluster.



(a) Geneva, epsilon = 1.1 (b) Geneva, epsilon = 1.5 (c) Geneva, epsilon = 2

Figure 4.2.: Comparison of Geneva DBSCAN epsilon values.

What is not very visible on the maps is that there are multiple data points for each location because of the 2 to 4 tweets that are generated for one historical hazard event. The accuracy test in Figure 4.4a shows us that epsilon 1.1 gives us only 0.59 accuracy. This means 59 out of 100 data points are clustered to a past data point. As the data points for the test are all taken from the old dataset it should be an accuracy of 1.

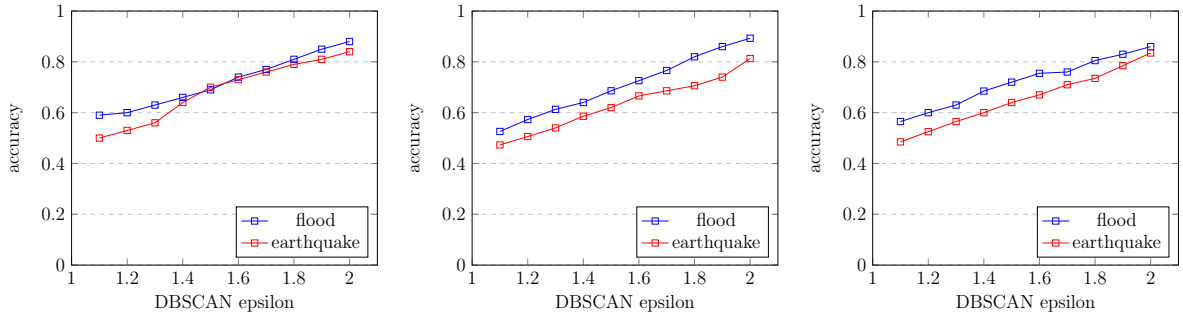


(a) Visp, epsilon = 1.1 (b) Visp, epsilon = 1.5 (c) Visp, epsilon = 2

Figure 4.3.: Comparison of Visp DBSCAN epsilon values.

The accuracy for epsilon 2 is 0.88, which is far better than that of epsilon 1.1. To consider the test with the maps and the accuracy test, it is proposed that we choose epsilon between 1.1 and 2. To still have an acceptable accuracy, it might be good to take epsilon higher than 1.5.

Comparing the accuracy across different numbers of data points N results in a decrease in the accuracy when N gets bigger. See Figure 4.4. This can suggest that the location, tags, and datetime configurations might need to be adapted dynamically.



(a) N=500, testN=100 (b) N=750, testN=150 (c) N=1000, testN=200

Figure 4.4.: Accuracy tests with location=10, tags=3, datetime=1

The procedure of the whole algorithm is as follows. It starts with the past data. Neighbors are calculated for the historical data, the adjacency matrices are created, and a logical OR is calculated on it to create one single adjacency matrix with all modalities fused. The result is an adjacency matrix of the relations between data points depending on the modalities.

In detail, the neighbors are calculated as follows: each data point is compared to the N-1 other data points in the chosen modalities. For each modality, a similarity method is chosen. See Table 4.3 for the different methods.

The modality "location" uses the haversine project [21] to calculate the distance between two data points with their latitude and longitude. The modality "tags" uses first a filtering over the tags. This filter operates tokenizing [37], stopword removing [36], stemming [35], and removal of own hazard terms. After filtering the set of tags, the tags are compared from one data point to the other with the cosine similarity. [13] The modality "datetime" uses the conversion of string to datetime object and calculates the absolute difference between data points.

modality	method
location	haversine
tags	tokenizing, stemming, stopword removal, hazard terms removal and cosine similarity
datetime	datetime object and absolute similarity

Table 4.3.: Modality similarity methods

The X nearest neighbors will be saved for each modality and each data point. This number is the k_m of the modality. For example, 10 nearest neighbors are needed for the modality location, and 3 are needed for the modality tags.

The adjacency matrices are then created by knowing the nearest neighbors for each modality and each data point. If another data point j is in the list of the nearest neighbors, the modality of data point i or the data point i is the same as j, the x_{ij} will be 1, else 0. This is done for every data point and results in adjacency matrices. See Algorithm 1 line 1. These matrices will then undergo a logical OR so that all the matrices (each standing for a modality) for one data point will be unified. See Algorithm 1 line 2.

After the old data is calculated, new data points can be calculated. Here is an example of how the algorithm transforms the data. If we say every modality takes one nearest neighbor, then this data will calculate the nearest neighbors and create the adjacency matrix.

```
data = [
              {"@id": 0,
              "tags": {"tag":["crue", "Eggiwil"]},
              "location": {
                        "@latitude": 46.8676734812958,
                        "@longitude": 7.81247335867426
                        },
              "@dateTaken": 2024-01-28 12:00:00
              },
              {
              "@id": 1,
              "tags": {"tag":["inondation", "Eggiwil"]},
              "location": {
                        "@latitude": 46.8676734812958,
                        "@longitude": 7.81247335867426
                        },
              "@dateTaken": 2024-01-30 12:00:00
              },
              {
              "@id": 2,
              "tags": {"tag":["crue", "Eiger"]},
              "location": {
                        "@latitude": 46.57840866962,
                        "@longitude":7.998180252154804
                        },
              "@dateTaken": 2024-01-31 12:00:00
              },
```

```
            {
            "@id": 3,
            "tags": {"tag":["inondation", "Eiger"]},
            "location": {
                        "@latitude": 46.57840866962,
                        "@longitude":7.998180252154804
                        },
            "@dateTaken": 2024-02-02 12:00:00
            },
      ]
```

Nearest neighbours were calculated for each data point and modality.

```
[
   {"location":[1],"tags":[1],"@dateTaken":[1]}, #id=0
   {"location":[0],"tags":[0],"@dateTaken":[2]}, #id=1
   {"location":[3],"tags":[3],"@dateTaken":[1]}, #id=2
   {"location":[2],"tags":[2],"@dateTaken":[2]} #id=3
]
```

Adjacency matrix for each data point and modality.

```
[
   [[1, 1, 0, 0], [0, 1, 1, 0], [0, 1, 1, 0], [0, 0, 1, 1]], #@dateTaken
   [[1, 1, 0, 0], [1, 1, 0, 0], [0, 0, 1, 1], [0, 0, 1, 1]], #location
   [[1, 1, 0, 0], [1, 1, 0, 0], [0, 0, 1, 1], [0, 0, 1, 1]] #tags
]
```

Unify the modalities of a data point into one list, which gives us the resulting adjacency matrix.

```
[
   [1, 1, 0, 0], #id=0
   [1, 1, 1, 0], #id=1
   [0, 1, 1, 1], #id=2
   [0, 0, 1, 1] #id=3
]
```

In the resulting adjacency matrix, we see the similarities between id 0 and id 1, id 1 and id 2, and id 2 and id 3. Whereas id 0 and id 3 are opposites.

The next step is the procedure for new data points. The goal is to identify new natural hazards in the new data. For the mode real, the scraper is activated to retrieve new data from Twitter. The scraped data (tweets) is transformed to have the same form as the old data. If the mode is set on test, there is a fixed dataset of new data points.

Each new data point will undergo the same process as the past data, which is to calculate the nearest neighbors, the adjacency matrices, and the logical OR to get one resulting adjacency matrix. This adjacency matrix created from the new data will be stacked with the past adjacency matrix, as seen in the below Code Listing 4.4.

```
adjacencyMatrix = np.vstack([self.adjacencyMatrix, adjacencyMatrix])
```

Code Listing 4.4: Stack new adjacency matrix

The difference between the process with the past data and the process with the new data is this next step: the calculation of the SVD and the feature matrix F. The N + 1 adjacency matrix will be inserted in an SVD. We reduce the dimension with D=100 and

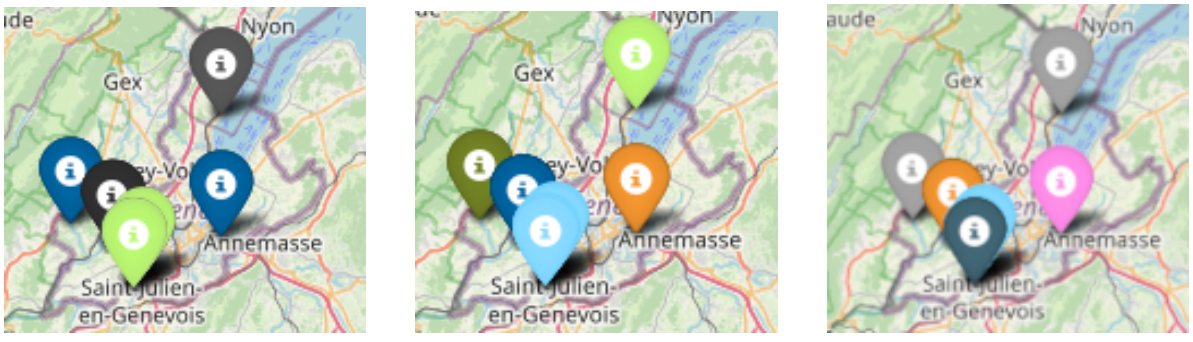
get u2, s2 and vh2. Sigma is then calculated with s2 and the reduced dimension D. To get the F feature matrix, we take the dot product of u2 and sigma. See Algorithm 1 lines 3 and 4.

To decide if a new data point is similar to an event that has already happened in the past, the k-means algorithm proposed in the paper, see Algorithm 1 line 5, is not used, but instead a DBSCAN. K-means has a fixed number of clusters, which does not completely suit the requirements. The DBSCAN was chosen because the number of events is not fixed. For example, we have 50 past events, and a new event will be either clustered to one of the 50 when it is a hazard or will give us a new label, which will mean that it is not a hazard. So we do not know, at the beginning of the algorithm, if there will be 50 or 51 events. Another advantage of DBSCAN is that it specifies the minimum number of data points for one label. Like this, it is possible to set it to one to enable a new label for single data points.

To compare the two methods, DBSCAN and k-means, an accuracy test and map comparison were done. As seen in Table 4.4, better accuracy occurs for the k-means when the number of clusters declines. If it clusters all data points to one label, the accuracy would also be 1, but this would label each new data point as a new hazard, which is not the goal. Therefore, the three numbers of clusters went through a map comparison. The same locations as in the DBSCAN map comparison were chosen for this comparison. For Geneva in Figure 4.5, the better result is achieved with c = 150. For Visp in Figure 4.6, there is no significance. With c = 100, too many data points are clustered together, whereas 150 clusters the dark blues together, which is irregular, and the c = 200 does not cluster them enough together. Based on this analysis and the possibility of setting the minimum data points of one label in the DBSCAN, it was decided to choose the DBSCAN as a clustering algorithm.

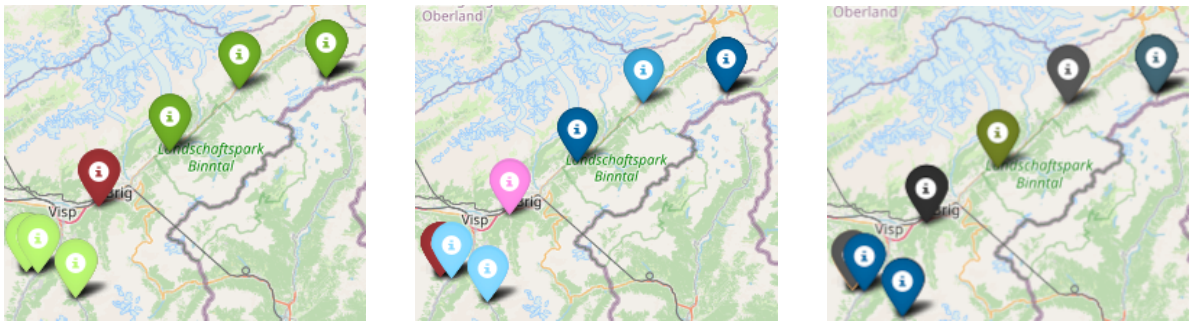
N	test N	loc	tags	date	nb clusters	ac test flood	ac test earthquake
500	100	10	3	1	200	0.75	0.69
500	100	10	3	1	150	0.91	0.93
500	100	10	3	1	100	1.00	0.98

Table 4.4.: K-means accuracy test



(a) Geneva, c = 100 (b) Geneva, c = 150 (c) Geneva, c = 200

Figure 4.5.: Comparison of Geneva k-means c values.



(a) Visp, c = 100 (b) Visp, c = 150 (c) Visp, c = 200

Figure 4.6.: Comparison of Visp k-means c values.

The DBSCAN is configured with parameters min_samples=1 and epsilon=1.75 for N=500. The minimum of samples per label is set to one so that events with only one tweet are also clustered as one label. After analyzing the data with maps and an accuracy test, the epsilon is set to 1.75. The F will fit the DBSCAN, and the labels will result. If the last element of the labels list is a new label, the new data point is classified as a "non-accepted" hazard. This means there is not enough evidence that this tweet is a hazard that should be alerted. Otherwise, the tweet will be classified as a hazard and set "accepted" to true.

The adjacency matrix will then be added to the past adjacency matrix in row and column so that it stays as a $N + 1 \times N + 1$ matrix and can be used to calculate further new data points without calling the database. See Code Listing 4.5.

```
1  import numpy as np
2  ...
3  self.adjacencyMatrix = np.vstack([self.adjacencyMatrix, adjacencyMatrix[-1]])
4  new_column = np.append(adjacencyMatrix[-1], 1)
5  self.adjacencyMatrix = np.column_stack([self.adjacencyMatrix, new_column])
```

Code Listing 4.5: New adjacency matrix

Once the classified new data points are obtained, they will be inserted into the database. Every data point is stored in the database, regardless of whether it was labelled as a hazard or not. This approach ensures that a new data point matching this non-hazard can still be correctly classified as a new event, which is defined by a new location, text, and date that has not happened previously. The data points labelled as hazard are assigned the value true and the others false for "accepted".

4.2.5. Scraper

The scraper explained in Chapter 3.1.2 is accessed from the backend program through a Python command. It is called in the run mode "real" to use real-life data. Each type of hazard runs the scraper on its own and sends its type to the scraper. The scraper returns a file per query. One query per keyword, for example, "flood" and "high water" will create a file with the scraped tweets. The scraped tweet files are then merged into one combined file. Next, each tweet is compared to a list of tweet IDs, which are saved after scraping to avoid inserting the same tweets into the algorithm as in a previous insertion

phase. New tweet IDs will be added to the list, and tweets that are already listed will be skipped.

The tweets scraped contain textual information about locations. This is transformed into numerical latitude and longitude data with the Website "Nominatim" [29]. It converts the location names or addresses into latitude and longitude coordinates. This conversion method can sometimes produce incorrect results. Errors may happen from ambiguous place names, misspelled inputs, or an incomplete database. In addition, the website has a limited access policy, so only one request can be made every second.

4.2.6. Database

To store the data a Postgresql database is chosen. Postgresql is an open-source relational database. The database consists of one table for the hazard reports, called hazard reports, and a second table for the scraped tweets, called tweets.

The first table saves the information about the hazard: its ID, the hazard type, the time and date the hazard happened, the latitude and longitude of its location, and the information "accepted" that defines if a new hazard matches with a past one and is therefore accepted by the algorithm "true" or not "false". The fields are all mandatory. The last field, "tweets," connects the tweet entries of the Tweet Model with the hazard.

The tweet table saves its ID, the type of hazard, the time and date of the posted tweet, the text of the tweet, the username of the publisher, and the latitude and longitude of the location where the tweet has been published. As mentioned in the chapter about Twitter scraping, the location might be falsely identified because if a tweet has no location information, the location is taken from the user's profile. The last field is to connect tweets to the hazard report. So a tweet has, at maximum, one report id stored. All fields are mandatory, except for the last one, in the case the tweet is not linked to a hazard report.

The reason two tables were chosen is that hazards can have multiple tweets. To save space, there will only be one entry per hazard and not for every tweet. The database is accessed through SQLAlchemy in Python.

4.3. Frontend Implementation

The frontend represents the part where the user interacts with the mobile application. It is implemented in Dart with Flutter [17] for mobile Android devices. The mobile application consists of four pages: a homepage, a tweet page, a history page, and a submit page. Each page appears with a symbol on the navigation bar at the bottom and is visible on every page. For each, a specific symbol was chosen.

The homepage in Figure 4.7a is the first page the user sees when opening the application. It shows a Google map that is implemented with the Google API. Switzerland is highlighted with a polygon with the data from GeoJson Maps [20]. Every alert for a new hazard will be shown on this map as a marker with its predestined color. These colors were chosen: blue for floods and red for earthquakes. Markers can be outside of Switzerland depending on the location data from the scraping. The only markers that do not appear outside of Switzerland are those inserted by users on the submit page. The markers are requested from the API with GET /api/alertData periodically. When

a new marker is found, a MarkerNotifier notifies the map to update the markers. The below code in Code Listing 4.6 shows how the listeners are notified when a new marker is inserted, which is not identical to others already displayed on the map. After the notification, the markers on the map are updated in Code Listing 4.7.

```
if (!_areMarkersEqual(newMarkers)) {
    _markers = newMarkers;
    print("notify markerListener");
    notifyListeners(); // Notify listeners if markers have changed
  }
```

Code Listing 4.6: Notifying marker listeners

```
final markerNotifier = Provider.of<MarkerNotifier>(context);
...

GoogleMap(
        markers: markerNotifier.markers,
        onMapCreated: _onMapCreated,
        initialCameraPosition: CameraPosition(
          target: _currentPosition,
          zoom: 7.0,
        ),
        polygons: polygon,
),
```

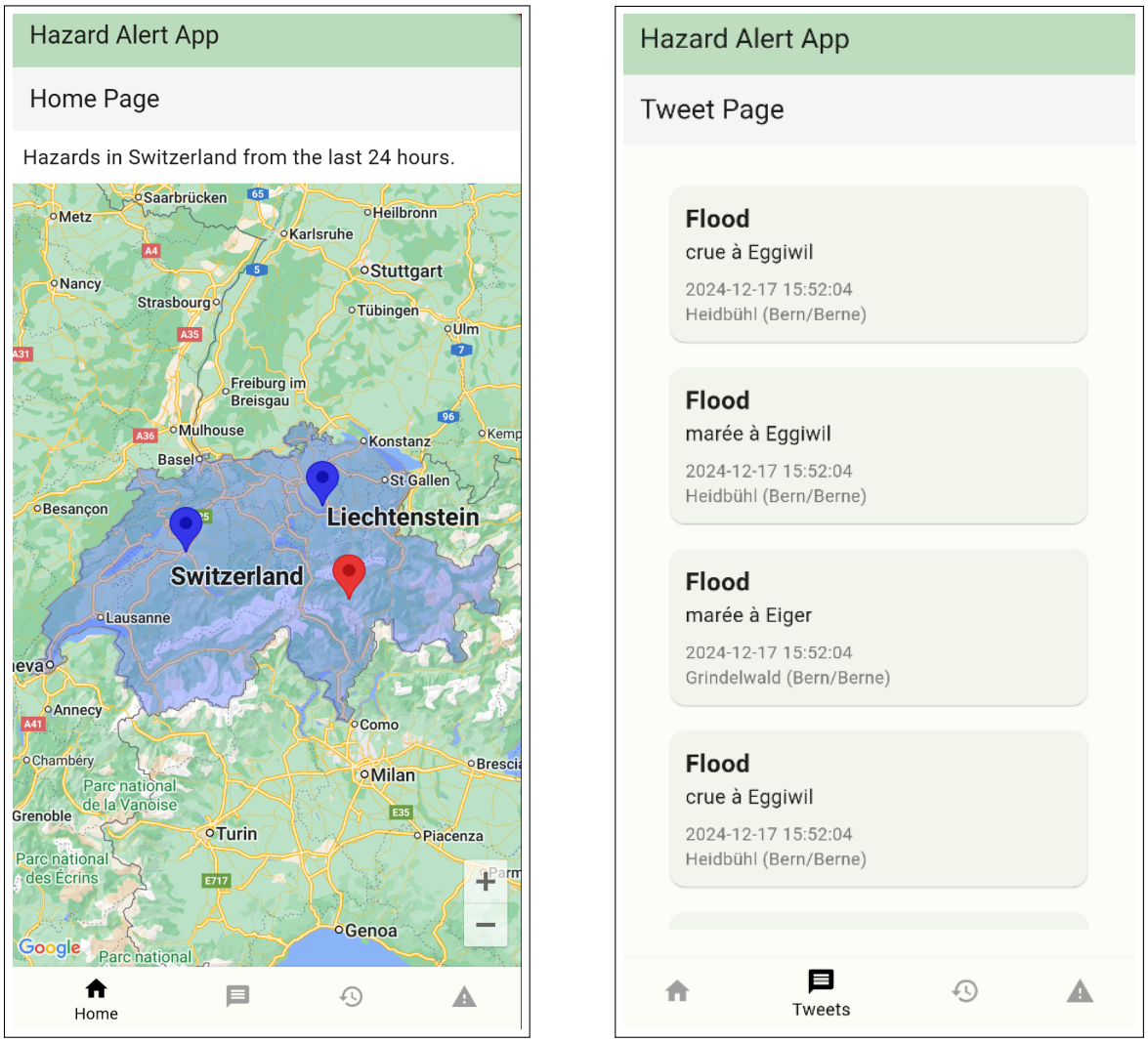
Code Listing 4.7: Marker updating

The tweet page in Figure 4.7b shows the tweet messages from the alerted hazards as a column of Flutter Card elements. It contains the hazard type, the date, the tweet text, and the location as text. If no alert has happened in the last 24 hours, no tweet cards will be displayed, and a message "No tweets available" will appear. If there are more tweets than fit on the page, the user can scroll up and down to see the others. The data is requested from the API with GET /api/tweetData.

The history page in Figure 4.8 shows a drop-down button to choose the type of hazard, a range slider for the selected years, and a bar chart. The bar chart displays the number of hazards of the chosen type per year and the range of years. It is possible to change the range of the years and the type of hazard. The data is requested from the API with GET /api/hazardsPerYear. This allows for the analysis of the number of hazards in recent years and the comparison of their development.

The submit page in Figure 4.9 has the goal of letting users report new hazards. A drop-down button lets the user choose the type of hazard, and on the Google map, the marker can be put to the location where the hazard is happening or has happened recently. The marker is only allowed to be put in Switzerland. This is achieved with a polygon with the data from GeoJson Maps [20]. To submit the report to the backend, a submit button is visible at the bottom. The submission is done with POST /api/data. All submitted hazards through this submit page will be marked as "true" for the accepted column in the database. This was chosen so that every hazard inserted by a user would be shown on the home page map.

A local notification service has been implemented to inform users of new hazard events.



(a) Home page (b) Tweet page

Figure 4.7.: Two pages of the application

Periodically, the API is called to determine the hazards from the last 24 hours. If a new hazard is part of the response, a listener is notified to generate the local alert notification for the user. The notification in Figure 4.10 then pops up, even if the app is not open now.

Two tasks are running to get notifications. If the user is on the homepage of the mobile application, it will request the backend for the hazard data points periodically. If not on the homepage but on other pages or outside the application, there is still a background task to find out if new events have been added. To avoid creating notifications for the already existing markers, preference storage is created, and the alerts are saved. When getting new markers on the homepage, the list of hazards is updated so there are not two notifications happening, one from the homepage and one from the background task. See Code Listing 4.8. When having a new marker, it will also be added to the preference storage. By polling, it compares periodically if a new marker was added. If it is the case the listener is notified to create a notification. See Code Listing 4.9.

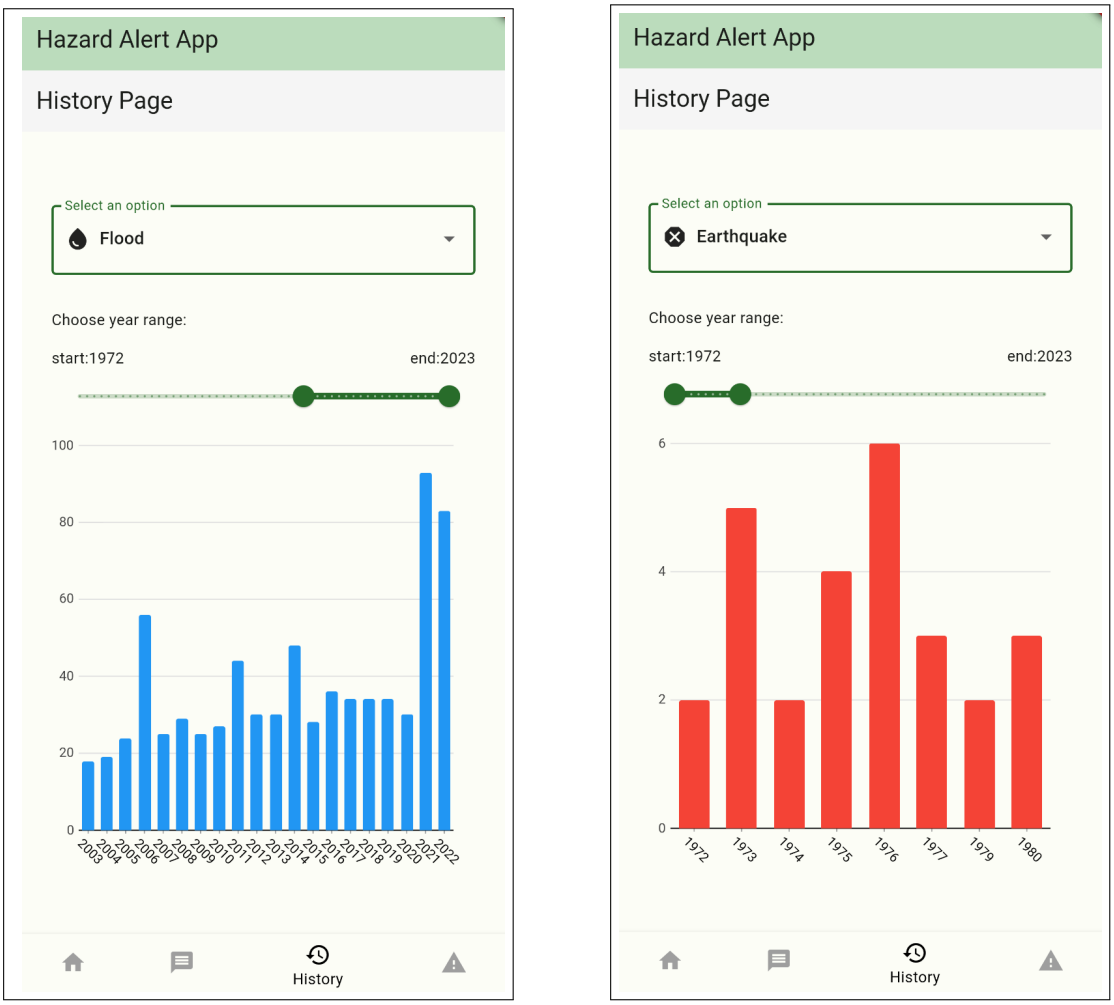


Figure 4.8.: History page

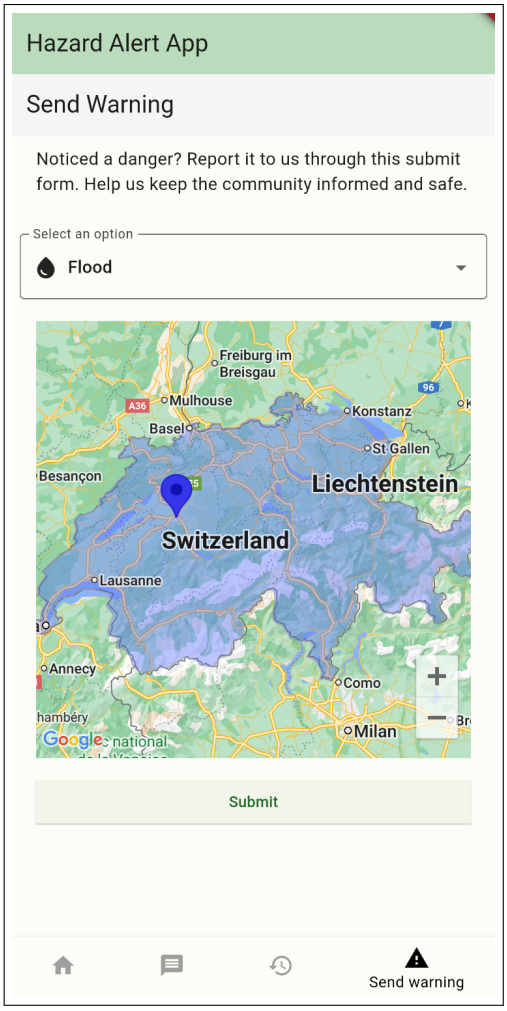


Figure 4.9.: Submit page

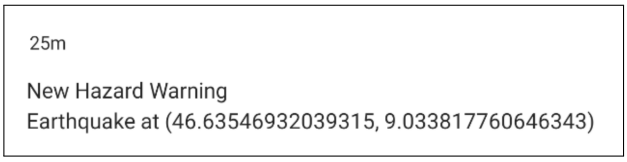


Figure 4.10.: New hazard warning notification

```
if (!alertIDs.contains(id) && first == null) {
            //set before so there aren't double notifications for background task and
                task on homescreen
            AlertValuesNotifier.getInstance()!.setAlertValues(newAlertIDs);
            print("notification");
            print(alertIDs);
            NotificationService().showNotification(title: 'New Hazard Warning', body: "
                $type at ($latitude, $logitude)");
          }
```

Code Listing 4.8: New event from database

```
void _startPolling() {
    _timer = Timer.periodic(Duration(seconds: 1), (timer) async {
      final prefs = await SharedPreferences.getInstance();

      final alerts = prefs.getStringList('lastAlerts');

      if (alerts != null && !_areAlertValuesEqual(alerts)) {
        _alertValues = alerts;

        notifyListeners();
      }
    });
  }
```

Code Listing 4.9: Polling function

4.4. Results with real data

Real-world tweets scraped from Twitter can have a wide range of various and complex text. The implemented program relies on filtering tweets with a specific timespan and keyword, for example, "flood". Therefore, tweets with metaphorical meanings are not excluded. As a result, a tweet using figurative meanings about, for example, political earthquakes or floods will still be inserted in the algorithm. If the modalities "location", "text" and "datetime" are similar to past data, it will be labelled as a natural hazard even if the post never mentioned anything about natural hazards and leads to false positives. This is a vulnerability of the application and needs to be addressed in the future. Implementing context recognition could improve the program and avoid false positives.

An example of such a tweet would be: "Erdbeben bei VP: Zürich-Chefin Mara Harvey weg ..." created on November 7 2024 at 9.27 am. The location is Zurich, taken from the users profile. In this case, the "Erdbeben" meaning earthquake is meant in a metaphorical way instead of speaking about a natural hazard.

4.5. Unit tests

Some unit tests were created for the implemented code. A unit test is a type of test to check smaller units of code for their correctness. For example, a function or a smaller

routine. [39] For this master thesis, tests for the backend and frontend are written. They especially test smaller parts and the logic of the application.

For the backend, unit tests were created with the package "unittest" [40]. Four test files have been created. To test the data fusion, two files are created: one where the calculation of neighbors is tested and one where the adjacency matrices and the final fused adjacency matrix are verified. For each test, an input is created and checked with a goal variable, which was calculated manually beforehand. For example, the adjacency matrix test is seen in Code Listing 4.10.

```
def test_calculate_adjacencymatrix(self):
      k_m = {"@dateTaken": 1, "location": 1, "tags": 1, "@username": 1}
      N = 4

      adjacencyMatrices = [[[1, 1, 0, 0], [1, 1, 0, 0], [1, 0, 1, 0], [1, 0, 0, 1]],
                           [[1, 1, 0, 0], [1, 1, 0, 0], [0, 0, 1, 1], [0, 0, 1, 1]],
                           [[1, 1, 0, 0], [1, 1, 0, 0], [0, 0, 1, 1], [0, 0, 1, 1]],
                           [[1, 1, 0, 0], [1, 1, 0, 0], [0, 0, 1, 1], [0, 0, 1, 1]]]

      adjacencyMatrix = AdjacencyMatrix.logicalOR(adjacencyMatrices, k_m, N)

      goal = [[1, 1, 0, 0], [1, 1, 0, 0], [1, 0, 1, 1], [1, 0, 1, 1]]

      self.assertEqual(goal, adjacencyMatrix)
```

Code Listing 4.10: Unit test adjacency matrix

For the clustering functions and the data transformations, such as transforming from scraped to algorithm data or from location name to latitude and longitude, separate test files were implemented.

In the frontend, unit tests were created with the package "flutter_test" [18]. A test example is the polygon, which is used to restrict input markers on the submission page for only Switzerland. For three scenarios, a test was implemented: data point inside the polygon, outside the polygon, and on the edge. They were tested with real latitude and longitude data points of Switzerland. In Code Listing 4.11, the test for the point inside Switzerland is shown.

```
  test('Point inside Switzerland polygon', () {
    LatLng pointInside = LatLng(46.505557, 7.952419);
    bool result = PolygonService.isInsidePolygon(pointInside);
    expect(result, true); // Expect true since the point is inside Switzerland
  });
```

Code Listing 4.11: Unit test polygon

Overall, more tests will be needed to ensure the application is correct. For example, to test the routing through API calls, the usage of the database and its storage, and user experiences. Tests like integration tests, end-to-end tests, and security tests can improve the application further.

5. Conclusion

In this master's thesis, a new disaster detection tool for Switzerland was implemented. The tool's distinctive feature lies in its multimodal approach, where tweets and numerical data are combined using a data fusion algorithm. With the knowledge gained throughout this thesis, the research questions were answered shortly, as follows.

How can social media data enrich numerical natural hazard data? Social media's multimodal data can provide supplementary context to the numerical data, such as text, videos, and images that depict specific natural hazard situations more vividly.

How can social media be used to provide effective alerts during natural disasters or environmental hazards? Social media posts, such as tweets about natural disasters, can facilitate faster responses to situations by offering comprehensible, real-time explanations of the situation rather than only having numerical data.

What are the key elements that encourage user interaction and engagement during natural hazards? A mobile application can encourage people to interact with it. Features such as a submission page allow people to insert their own alerts about situations they are experiencing and thus help others be alerted in time.

What are the fundamental components and best practices for designing a software architecture for a hazard detection system? Key components of this implementation are Twitter scraping (or access to the Twitter API), a backend system with data fusion and clustering, a database storing the multimodal data, and a front-end mobile application for user interaction.

Throughout the research for this master's thesis, I have gained valuable experience in developing complete software with different components like the Twitter scraper, the database, the entire algorithm with data fusion and clustering, and the creation of a mobile application. There were challenges like collecting data from different institutions or the Twitter API restrictions.

The implementation of the disaster detection system highlighted potential improvements for future work. As the database grows over time, it will be essential to analyze if the chosen configurations for the number of neighbors per modality must be changed at some point to get good results and if they should be adapted dynamically depending on the size of the database. Additionally, the selection of the modalities could be analyzed and expanded. In the future, images and videos could also improve the application, and the analysis of the weighting of the modalities will be necessary to identify optimal and accurate results.

Furthermore, the current Twitter scraper filters tweets with keywords. This presents a limitation, as it may also scrape tweets unrelated to natural hazards but with a metaphorical context. It gives the opportunity to analyze it and to refine the process. Additionally, the location of the tweets is currently scraped via tweet and user profile. This must be further evaluated to see if the profile gives accurate results. Another aspect is the login to Twitter. As Twitter can recognize activities, the user might be blocked. In the future,

another solution may be found.

Lastly, the user-friendliness of the mobile application. Running user tests will help improve the functionality and usability of the application. Additionally, the submission form must be analysed and a technique must be implemented, to verify the accuracy and validity of the submission input. This could help prevent misuse of reporting alerts.

A. Common Acronyms

SVD Singular Value Decomposition
DBSCAN Density-Based Spatial Clustering of Applications with Noise

B. GitHub repositories of the mobile application

A mobile application was created for this master thesis and these are the links to the private GitHub repositories:

- https://github.com/ghoussod/hazardalertapp_backend
- https://github.com/ghoussod/hazardalertapp_frontend

References

[1] Imad Afyouni, Zaher Al Aghbari, and Reshma Abdul Razack. Multi-feature, multi-modal, and multi-source social event detection: A comprehensive survey. *Information Fusion*, 79:279–308, 2022. 10

[2] Prasanna Giridhar, Shiguang Wang, Tarek Abdelzaher, Tanvir Al Amin, and Lance Kaplan. Social fusion: Integrating twitter and instagram for event monitoring. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 1–10. IEEE, 2017. 4

[3] Rui Li, Kin Hou Lei, Ravi Khadiwala, and Kevin Chen-Chuan Chang. Tedas: A twitter-based event detection and analysis system. In *2012 IEEE 28Th international conference on data engineering*, pages 1273–1276. IEEE, 2012. 3

[4] Yun Ma, Qing Li, Zhenguo Yang, Zheng Lu, Haiwei Pan, and Antoni B Chan. An svd-based multimodal clustering method for social event detection. In *2015 31st IEEE International Conference on Data Engineering Workshops*, pages 202–209. IEEE, 2015. 9, 10

[5] Amitangshu Pal, Junbo Wang, Yilang Wu, Krishna Kant, Zhi Liu, and Kento Sato. Social media driven big data analysis for disaster situation awareness: A tutorial. *IEEE Transactions on Big Data*, 9(1):1–21, 2023. 7

[6] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, pages 851–860, 2010. 3, 5, 8

[7] Jianshu Weng and Bu-Sung Lee. Event detection in twitter. In *Proceedings of the international aaai conference on web and social media*, volume 5, pages 401–408, 2011. 3

[8] Chaolun Xia, Raz Schwartz, Ke Xie, Adam Krebs, Andrew Langdon, Jeremy Ting, and Mor Naaman. Citybeat: Real-time social media visualization of hyper-local city data. In *Proceedings of the 23rd international conference on world wide web*, pages 167–170, 2014. 3, 5

[9] Chao Zhang, Guangyu Zhou, Quan Yuan, Honglei Zhuang, Yu Zheng, Lance Kaplan, Shaowen Wang, and Jiawei Han. Geoburst: Real-time local event detection in geo-tagged tweet streams. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 513–522, 2016. 4

Referenced Web Resources

[10] Website of AlertSwiss. https://www.alert.swiss/de/app.html (accessed October 24, 2024). 4

[11] Website of AlertSwiss FAQ. https://www.alert.swiss/en/faq.html (accessed October 30, 2024). 4

[12] Website of BAFU natural hazards. https://www.bafu.admin.ch/bafu/en/home/topics/natural-hazards/in-brief.html (accessed November 10, 2024). 6

[13] Website of GeeksforGeeks: Cosine Similarity. https://www.geeksforgeeks.org/cosine-similarity/ (accessed November 03, 2024). 19

[14] Website of Natural Hazards: Dangel levels . https://www.natural-hazards.ch/home/dealing-with-natural-hazards/earthquakes/danger-levels.html (accessed November 22, 2024).

[15] Website of Docker. https://www.docker.com/ (accessed October 24, 2024). 15

[16] Website of Flask. https://flask.palletsprojects.com/en/3.0.x/ (accessed October 24, 2024). 16

[17] Website of Flutter. https://flutter.dev/ (accessed October 24, 2024). 2, 24

[18] Website of Flutter: flutter_test. https://api.flutter.dev/flutter/flutter_test/flutter_test-library.html (accessed November 24, 2024). 30

[19] Website of PyPI Folium. https://pypi.org/project/folium/ (accessed November 15, 2024). 17

[20] Website of GeoJSON maps. https://geojson-maps.kyd.au/ (accessed November 03, 2024). 24, 25

[21] Website of Haversine pyPI. https://pypi.org/project/haversine/ (accessed November 03, 2024). 19

[22] Website of Map Developers: Draw a circle - Create a circle on a google map using a point and a radius . https://www.mapdevelopers.com/draw-circle-tool.php (accessed November 21, 2024). 8

[23] Website of Meteo Swiss: Hazard map. https://www.meteoswiss.admin.ch/weather/hazards/hazard-map.html (accessed October 30, 2024). 4

[24] Website of Meteo Swiss: Natural hazards map. https://www.meteoswiss.admin.ch/services-and-publications/applications/hazards.html (accessed October 24, 2024). 4

[25] Website of Meteo Swiss: How severe-weather warnings are prepared. https://www.meteoswiss.admin.ch/weather/hazards/how-severe-weather-warnings-are-prepared.html (accessed October 30, 2024). 4

[26] Website of Multimodal Machine Learning: Data Fusion

published in Towards AI. https://pub.towardsai.net/multimodal-machine-learning-data-fusion-d1d8776e2cb0 (accessed November 08, 2024). vi, 10, 11

[27] Website of Medium: Multimodal Models and Fusion - A Complete Guide. https://medium.com/@raj.pulapakura/multimodal-models-and-fusion-a-complete-guide-225ca91f6861 (accessed November 24, 2024). 10

[28] Website of NCCS WSL. https://www.nccs.admin.ch/nccs/en/home/the-nccs/about-the-nccs/organisation/members-and-partners/swiss-federal-institute-for-forest--snow-and-landscape-research-WSL.html (accessed November 10, 2024). 6

[29] Website of Nominatim: Search . https://nominatim.openstreetmap.org/search (accessed November 24, 2024). 24

[30] Website of Postgresql. https://www.postgresql.org/ (accessed October 24, 2024). 2

[31] Website of SED ETH Zurich. http://www.seismo.ethz.ch/en/home/ (accessed November 10, 2024). 6

[32] Website of SED ETH Zurich earthquakes Switzerland. http://www.seismo.ethz.ch/en/earthquakes/switzerland/all-earthquakes/ (accessed November 10, 2024). 6

[33] Website of Selenium. https://www.selenium.dev/ (accessed October 24, 2024). 2

[34] GitHub Repository of godkingjay's Selenium Twitter Scraper. https://github.com/godkingjay/selenium-twitter-scraper (accessed November 17, 2024). 7

[35] Website of stemming NLTK. https://www.nltk.org/howto/stem.html (accessed November 03, 2024). 19

[36] Website of stopwords NLTK. https://pythonspot.com/nltk-stop-words/ (accessed November 03, 2024). 19

[37] Website of tokenize NLTK. https://www.nltk.org/api/nltk.tokenize.html (accessed November 03, 2024). 19

[38] Twitter Website Advanced Filtering for Geo Data. https://developer.x.com/en/docs/tutorials/advanced-filtering-for-geo-data (accessed November 17, 2024). 8

[39] Website of SmartBear: What is unit testing. https://smartbear.com/learn/automated-testing/what-is-unit-testing/ (accessed November 24, 2024). 30

[40] Website of Python Docs: unittest. https://docs.python.org/3/library/unittest.html (accessed November 24, 2024). 30

[41] Website of WSL's flood and landslide damage database. https://www.wsl.ch/en/natural-hazards/understanding-and-forecasting-floods/flood-and-landslide-damage-database/ (accessed November 10, 2024).