Wisdom of the Local Crowd

Detecting Local Events Using Social Media Data

sw703e15
Lasse Drustrup Christensen
Lukas Nic Dalgaard
Mads Ellersgaard Kalør
Sofie Aaskov Nielsen
Søren Bøtker Ranneries

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Selma Lagerlöfs Vej 300
DK-9220 Aalborg Ø
http://www.cs.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Wisdom of the Local Crowd: Detecting Local Events Using Social Media Data

Theme:
Internet Technologies

Project period:
Autumn semester 2015

Project group:
sw703e15

Participants:
Lasse Drustrup Christensen
Lukas Nic Dalgaard
Mads Ellersgaard Kalør
Sofie Aaskov Nielsen
Søren Bøtker Ranneries

Supervisor:
Nattiya Kanhabua

Copies: 7

Pages: 70

Date of completion:
December 21, 2015

Abstract:

Social media contain vast amounts of user generated information describing experiences from users' daily lives. We propose a method for extracting events from this data with the purpose of providing a list of ongoing events that are interesting to participate in. We gather posts from Twitter and Instagram, and perform a number of processing steps to identify posts related to events. We use hashtags and location information to cluster similar posts and a classifier to identify groups of posts which represent events. We contextualize events by estimating their sentiments and categories, and by extracting descriptive keywords. We present a web application for displaying events to users so as to explore possible representations of events. Our experiments show that the system is able to detect events with an F_1 score of 0.2553. We have reason to believe that the score can be improved significantly by using more training data and adjusting the hyperparameters.

Preface

This report is conducted by five Software Engineering students during the 1st semester of their masters education. The semester takes place at the Aalborg University from September 2, 2015 to December 21, 2015. The overall theme for this project, supplied by the university, is “Internet Technology”.

Lasse Drustrup Christensen

Lukas Nic Dalgaard

Mads Ellersgaard Kalør

Sofie Aaskov Nielsen

Søren Bøtker Ranneries

Contents

1 Introduction

The desire to go out often arises spontaneously. When this happens, and no preexisting plans have been made, one must figure out what is happening now or in the immediate future. There are several ways to find out what is going on. One can for example ask friends, check the calendar of venues, or look at event aggregation sites — but no single source has an exhaustive list of events happening right now near one's current location. To get a complete overview, one has to look through several sources of information. An approach to finding out what one's friends are doing is by looking at social media.

Social media is popular all over the world. Every day, millions of people share information about experiences from their daily lives with their friends and the world[28]. Facebook had over one billion active users during September 2015[43], and these users generate large amounts of information on a variety of different topics.

People like to share experiences with each other. This is also true for users of social media. They are likely to post pictures or a short status message whenever they are at events like concerts or football matches. If an event has a sufficient number of participants, it is almost given that someone will make a post about that event on social media. Around the world these posts can be used to provide an insight into local happenings. However, extracting information about events from a stream of posts generated by the users of social media is a large and complex task. The data is unstructured and noisy, so to accomplish this task, several smaller challenges must be overcome. The content of the posts must be analyzed to determine if they describe an event, and if they do, it must be found which event they describe. In addition, the sheer amount of data generated by people on social media demands an automated approach to solving this challenge.

In this report, we present an event detection system which uses streams of posts from social media to detect and analyze events such that the users of the system are able to make an informed decision on whether they wish to attend them.

1.1 Users

In general, many different types of people and organizations could be interested in finding ongoing local events with different motivations. Families could be looking for entertainment offers nearby and young people could be looking for parties to participate in. A police department could be interested in knowing about events happening right now in their jurisdiction, as to better react to spontaneous gatherings of people. A shop owner could capitalize on a nearby event as to attract a crowd of people. For this project, we put our focus on a specific group, namely tourists looking for nearby events while traveling. This allows us to design the user interface with them in mind.

According to [22], tourists can be described by three parameters based on how they prepare for a trip: *Search*, *plan*, and *bookings*. *Search* refers to research of attractions in a city, *plan* refers to deciding for what to do before the trip, and *bookings* refers to the degree in which people book tickets for attractions, restaurants, and hotels beforehand. Tourists can be separated into four segments, depending on how they prepare for a trip:

1. Low search, low plan, and low bookings
2. High search, low plan, and low bookings
3. High search, high plan, and low bookings
4. High search, high plan, and high bookings

The first group prefer spontaneous planning. The second group prefer to know about a city before they go, but they make no strict plans. The third group research and plan ahead of a trip, but they still leave a little freedom in the specific places to go. The fourth group represents people who prepare a lot for a trip and generally want a precise plan of what to do. Overall, young people tend to be more spontaneous in their travel planning than elderly people [22].

The first two segments of tourists represent the target of our service, as they generally have few plans for their trips. They are likely interested in joining events they discover during the trip if they find them interesting.

1.2 Current Approaches

When a young tourists want information about what is happening right now, there are different means to approach the problem. One approach is to visit the local tourist agency and others are to find events on different kinds of events related websites. In this section, we look into some existing information providers and evaluate their respective advantages and disadvantages. All the mentioned providers represent different approaches and can be seen as examples of these.

Tourist Agency Most large towns have their own tourist agency with both a physical location and a web pages where interested people can get information about events.

The VisitDenmark web site[1] has events which are happening now or in near future. The events are large and they are all planned in advance, and they have been manually selected by editors.

Time Out Time Out[2] started out as a magazine which wrote about culture, entertainment, and events in London. Today, Time Out consists of a web page and apps for the most popular mobile devices, and covers a broad spectrum of events in 89 different cities all over the world. Their web page and apps make it possible to locate events in the cities they cover. The events are grouped either by date or by type, e.g. concerts and conferences. The events are created and managed by editors which means that spontaneous events are not present. All events they present are public.

Facebook Groups On Facebook multiple sites where users can post events exist. One is the site called "Whats Happening in East London". The purpose of sites like this is to allow users to find or share upcoming or ongoing events in their local area. Unfortunately, these sites do not have efficient filtering and spam detection which means that they contain unrelated posts. For example, the aforementioned London group receives up to hundreds of posts every day, and a great deal of these are spam[14].

Facebook Event Recommendation Facebook has four ways of recommending events to a user[16]. Those are events friends participate in, events which are similar to events the user has previously participated in, events created by sites the user likes, and popular events nearby the user. All these events are public and manually created by users. Facebook present the events in a presorted way based on what they think is relevant for a specific user, and there are no filtering or sorting options for the events which can make them difficult to manage.

Swarm Swarm is an application with the purpose of enabling meet-ups with friends. Its main access point is a mobile application where you connect with friends and have the ability to "check in" at a location, which could be where an event is taking place. It is possible to see where one's friends have checked in within a given distance, which may represent events they are participating in. These events may be planned or spontaneous. However, Swarm only shows activities that one's friends participate in[39].

Young tourists could make good use of services like Time Out or tourist agencies. These services make good recommendations of events but they are limited by the fact that they people have to manually find, describe, and create the events on their websites. This would potentially not provide tourist with small or spontaneous events happening near them. Such events could be provided by Facebook, but the management Facebook performs of its "Events near you" event recommendations is unstructured and a user has no way of filtering or searching the events. One could also use user-created Facebook

[1] www.visitdenmark.com
[2] www.timeout.com

groups for events at a given location. However, due to the lack of filtering and spam detection, there are many unrelated posts and no way of filtering them. All of this is likely to make it hard for e.g. a tourist to find genuine events on Facebook using their groups or event recommendation. Swarm is only useful as long as the user is near people they know, since Swarm only shows check-ins by friends.

From the shortcomings and strengths of existing providers, we form a list of focus areas that a new approach should take into account:

- Automated event detection so no or little manual creation of events is needed.
- Location based recommendation so the user can see what is going on nearby.
- Give the user an option to search, filter, and sort events to make the solution user friendly.
- Limit, if not avoid, unrelated material and spam to make it the events reliable and the system user friendly.

To comply with the first focus area, we find a digital approach necessary. We argue for gathering data from social media in order to find events as it allows for compliance in all four areas. Users on various networks are already sharing content, so we see opportunities in using these information sources, can we separate posts related to events from the rest.

1.3 Social Media

In this section we compare the most common social media and evaluate them based on the three criteria we consider essential for discovering happenings in prioritized order: access to data, relevance of content, and real time posting.

We use the following definition of social media introduced by [30]: “Social Media is a group of Internet-based applications that build on the ideological and technological foundations of Web 2.0, and that allow the creation and exchange of User Generated Content”. Web 2.0 refers to the era in which content is largely generated by users rather than individuals. For our purposes we will not consider social media where the content shared is private, or the main language is not English.

For a social medium to be relevant for this project it must have many active users providing recent data from many locations. Mobile device users are more likely to share content during an event. Information including tags are advantageous, as they can provide the topic and possibly the mood of the content. It is strictly necessary that the data from the social media can be accessed either by crawling or through an Application Programming Interface (API).

1.3.1 Comparison of Social Media

In the following, we describe the most popular social media with public content used in the Western world.

Facebook Facebook is the largest network and has 1490 million active users every month[52]. However, Facebook does not allow any kind of crawling or data collection unless each individual user explicitly allows it[15].

Tumblr Tumblr is a blogging web site with 230 million users[52]. As a blogging site, event content is more likely shared after an event has happened.

LinkedIn LinkedIn is a business oriented social medium with 97 million month users[52].

Google+ Google+ is a network with 300 million monthly users[52]. However, only around 6.6% of the users on Google+ log in via mobile devices[51]. Google+ is centered around communities, which connect people through interests instead of friends.

Twitter Twitter has a user base of approximately 300 million[52]. The most notable thing about Twitter is the user activity with 550 million tweets per day[44]. Twitter has an API which makes their data readily available[55].

Instagram Instagram is a picture sharing social media network. It is only possible to post pictures via a mobile platform[24]. Instagram has a user base of approximately 300 million[52]. Instagram users make more use of geo-location tags than Twitter users [58]. The user base primarily consists of young people[11]. It is also allowed to crawl their data, though they also have an API[25].

We eliminate Google+ as its mobile user base is a small part of their already limited user size. To get access to user information from Facebook, each user has to allow our application to access their information. As we do not want to make an effort to get users on Facebook to sign up for this application, we will not use Facebook as a data source. LinkedIn is focused on business relations, rather than social, so it is also not suited. Tumblr users are unlikely to post relevant content in time for it to be of use, since it is not likely for one to blog during an event.

Twitter and Instagram are suitable sources of data. Their data is available, the users often add tags to describe the topics of the posts, and have the greatest mobile device usage. They are dissimilar enough in content to supplement each other. Therefore, we choose to use a combination of both social media as a data source.

1.3.2 Event Characteristics

To understand how events are present on social media, we explore Twitter and Instagram in order to identify characteristics of events. We find that these media contain information about multiple ongoing events, one of which being the London Design Festival. Posts about London Design Festival are present on both Twitter and Instagram and are often labeled with the same hashtags, e.g. “#Londondesignfestival” and “#ldf15”. Additionally, at least 10 of the posts have pictures and an explicit location of Covent Garden, which is one of the places where the festival takes place. We find that hashtags related to events are used prior to, during, and after the event for promotional purposes. Some hashtags

are not directly associated with any specific event but rather used for special kinds of events. One such hashtag is "#livemusic", which seems to always indicate an event but often different events.

In general, hashtags are widely used on social media. On Twitter, hashtags are often used to denote the topic of the post or to connect the post to a larger ongoing discussion[21]. This is also expressed by the presence of event-specific hashtags, which are often created by the event organizer to ensure that the same hashtag is used for an event. The wide use of hashtags as event identifiers makes hashtags suited for determining whether posts are related to an event or not.

1.4 Problem Statement

We use the definition of an event given by Yang et al. [57]: "An event identifies something (non-trivial) happening in a certain place at a certain time". By non-trivial, we mean that the event must have some significance. An event is not significant if it happens often, e.g. a train arriving at a platform. An event can be trivial even though it is rare, for instance, a song being played out of the speakers on a bar is considered trivial. Conversely, an event can also be non-trivial, even though it takes place for several months; special exhibitions in a museum are considered as non-trivial. The distinction can be quite small at times. A football match in a stadium is a non-trivial event, but the same match shown on a television in a bar is a trivial event, unless people have gathered at the bar with the purpose of watching the game.

In this project, we seek to answer the following question: *How can ongoing geospatial events be detected early from Twitter and Instagram streams?* This question can be separated into several smaller and more specific questions:

- How can events be detected early, so that they can be discovered while they are ongoing?
- How can events be distinguished from non-event data such as breaking news?
- How can events be presented such that young tourists can find those they find interesting?

1.5 Related Work

Event detection on social media is a widely researched area of information retrieval. Some work is related to detection of large-scale events such as earthquakes [47] and president election prediction [54]. While this is useful for certain types of events, it differs from local event detection in the amount of data available. There are several reasons for this. One is that the number of users interested in large-scale events is greater than those interested in local events. Another is that posts need not be associated with a specific location, e.g. the sentiment of any English post about the national election can potentially be used to predict election results.

Some research focuses on detection of geospatial event detection. One approach to local event detection is presented by Becker, Naaman, and Gravano [5]. They describe a system for detecting real-world events in a Twitter stream based on tweets from New York City. To generalize from this location, however, locations need to be manually defined. Their approach is largely based on clustering of tweets. They cluster tweets based on their topics and temporal rates and classify whether each cluster is a real-world event or not. They do not assign a location to each event. We work upon their approach of classifying clusters rather than individual posts as events and non-events. However, topical clustering alone is not sufficient when we seek to associate a location to each event, since a certain topic may be present in multiple world wide events. In addition, the geographical resolution is insufficient for our problem.

Walther and Kaisser [56] approach the subject of detecting events by first clustering based on location then applying a Machine learning component to determine if a cluster is an event or not. Compared to the work of Becker, Naaman, and Gravano [5], this approach gives higher location granularity, and we work upon the idea of location clustering. They identify 41 different features that can be used for classifying clusters as events and non-events. Only geotagged posts are considered, and the approach does not allow several events to happen on the same geographical location. Taking the inaccuracies of mobile GPS devices into account, this seems unfortunate for local event detection, since several events are likely not clearly separated geographically. We attempt to solve this shortcoming by performing both location and topical clustering.

A somewhat different approach is that of Lee and Sumiya [34]. They attempt to detect geographic events by identifying local irregularities in the rate of tweets, the number of unique users, and the movement of users. They divide Japan into several *Regions-of-Interests* determined by a K-means clustering of geotagged tweets (with an empirically defined number of regions). Their work shows promising results in detection of large-scale events such as festivals and earthquakes. However, their approach of dividing the earth into regions-of-interest requires manual tuning, and the system is unable to detect small-scale events such as performances by street musicians.

All mentioned work is solely based on Twitter data. Other social media platforms such as Instagram contribute with valuable data for local event detection. For example, a study shows that five times as many Instagram posts than Twitter posts are geotagged [58]. To generalize from Twitter, we present a system that has low coupling between social medium post format and the event detection system.

1.6 Overview of Approach

The following chapters will describe the event detection system in detail. Figure 1.1 shows a conceptual illustration of the different parts. People at events publish statuses on social media about the events they are attending. The data is preprocessed before it is analyzed in order to identify events. These events are contextualized such that they can be presented informatively in a web application.

This report is structured as follows. In Chapter 2, we describe the requirements for

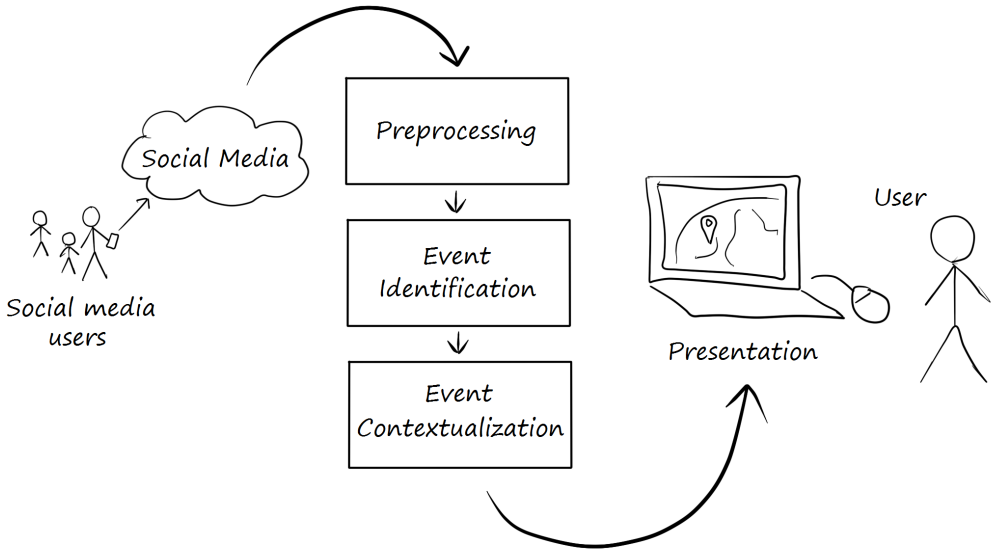


Figure 1.1: Conceptual illustration of the event detection approach

this solution and present the user interface design. In Chapter 3, we present the theory we apply in order to fulfill the problem statement. Following is Chapter 4, in which we describe how we collect data from social media and process it in order to detect ongoing events. This chapter also presents how these events are made presentable to the users by contextualizing the events with semantic properties. Following, in Chapter 5, we evaluate and discuss the event detection and contextualization. In Chapter 6, we describe how we present the detected events through a web application to the users. In Chapter 7, we conclude on the outcome of the project and suggest which parts of the system future work should focus on. Finally, in Chapter 8, we reflect upon the development method used throughout the preparation of this project.

2 Requirements and Design

In this chapter, we describe a requirement elicitation resulting in a set of functional requirements. Afterwards, we present a navigation design describing a user interaction with system which fulfill the design specific requirements. Lastly, we present a user interface based on the described navigation.

2.1 Requirements

As previously mentioned we want to create a system which can find ongoing events on social media and represent it through a web application. Before designing the system we specify some requirements for the system. We use the goal analysis model called *i**, which makes use of the entities: actors, goals, tasks, resources, and dependencies[8]. The i* model that our system is based on, seen in Figure 2.1, focuses on points from the current approach as described in Section 1.2.

The goal for the system is to satisfy the users by automatically finding interesting events for them to participate in. In order to do so, the system first needs to extract events from posts on social media. The extracted events must all be ongoing events. In addition, the system must present the events so that the user is able to determine if they would find the event interesting and whether it is within a reasonable distance.

Before the user is able to determine if an event is interesting, they need some information describing the event. The work by [37] shows that pictures benefit in the summarization of events. In addition to pictures, we choose to label the events with additional measurements: size, category, and sentiment. Size indicates how many people there are at the event, as some users may consider that an important factor. Sentiment specifies the atmosphere or mood of an event, as a rough estimation of how other people react to the event. Category tells the user whether an event is about e.g. sport or art, as different users have different fields of interest. Lastly, the event description must be supplemented with more detailed information, by showing a number of relevant keywords.

On this basis, the following functional requirements describe the necessary behaviors of the system, and they serve as guidelines for the development of the system:

- Extract current events from social media data
- Find the location of an event
- Find pictures of an event
- Find the size of an event
- Find the sentiment of an event

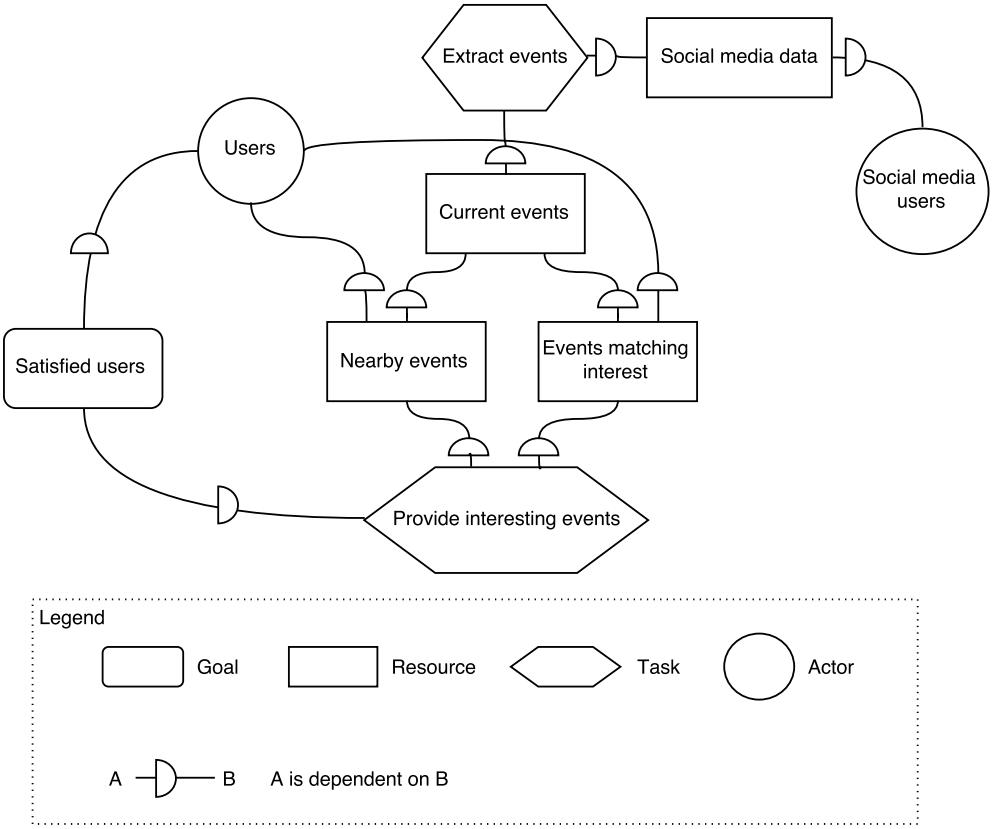


Figure 2.1: Goal model of the system

- Find the category an event belongs to
- Find keywords to describe an event
- Get a user's location
- Get a user's interests
- Display the size, location, sentiment, category, keywords, and pictures associated with an event

2.2 Navigation Design

The navigation design of the web application is based on a workflow chart. Figure 2.2 illustrates a workflow where a user wants to find relevant events.

The workflow begins when the user opens the application. As long as the application is open, the user has the option of closing the application and can receive the error message indicating that no events are found. Upon startup, the application finds the

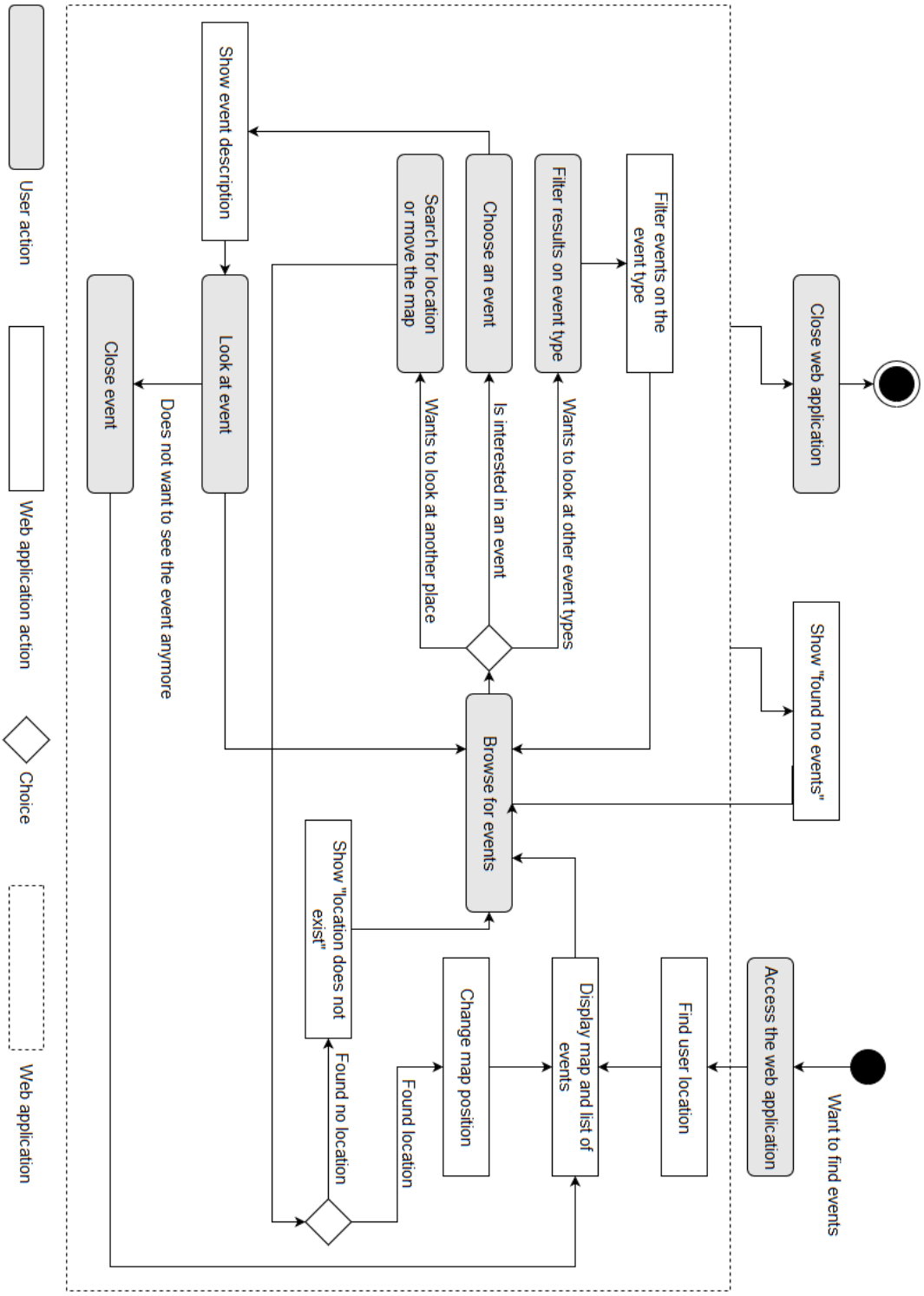


Figure 2.2: Workflow chart of the web application. A solid circle represents the start state and a solid circle with a hollow circle outside represents the final state.

user's location and centers the map it on. This is done so the user easily can get started on finding events. As the application locates the user it also updates a list with events, which are located within the visible part of the map shown to the user.

We decide to use the faceted navigation method to provide the user with different tools for browsing events. These are: filter the list of events on category, sentiment, or size; select an event; search for a location; or navigate the map manually. When the user applies filters, the application will update the view to show only the relevant events. If the user searches for a location, by entering a location in a search field, two things can happen. Either the location is found, which results in the map view changing to have the location in the center and the event list will update. Otherwise, the search fails, which results in an error message indicating that no matches were found. Alternatively the user can move the map manually to find a location, this will never generate an error message.

The user can choose an event from the list of events. When doing so the application will show relevant data for that particular event, such as precise location details, a description, and pictures. While the user has an event selected, the user cannot see the list of events but can still filter, search, and navigate the map in general. When the event is deselected, the user will return to the list view, which is updated with whatever changes have been made while looking at the event.

We expect this design to satisfy the last three requirements stated in Section 2.1. The users can express their interests by applying filters. Selecting an event will display details associated with the event, such as location.

2.3 User Interface Design

The main focus for this prototype is the event detection. Therefore, the user interface is designed for the desktop platform only. We choose to limit the scope to desktop computers since it is less time consuming to design and implement an interface for a desktop screen than for a smartphone screen. At this stage of the prototype the main focus of the interface is to find a good representation of the events, and a desktop interface will be sufficient for this purpose.

The user interface is based on the workflow diagram in Figure 2.2. The design of the web application is inspired by the Google maps design as it should be familiar to the users. Figure 2.3 is a sketch of the web application, which illustrates the five components of the interface: map, search bar, filtering bar, zoom buttons and a sidebar. As we want the main focus of a user to be on the map; the other components hover over the map. The bars align to the left and are sized to $\frac{1}{3}$ the width of the page, though with some minimum width. This leaves the remaining $\frac{2}{3}$ of the page to the map.

Following we describe each individual element of the interface:

Map The map is the main element. The map gives the user a visual representation of where the events are taking place. When the application is opened, it should be centered on the user's location and it should be highlighted. Events are represented by markers. The if the user click on a marker, the event description appears in the

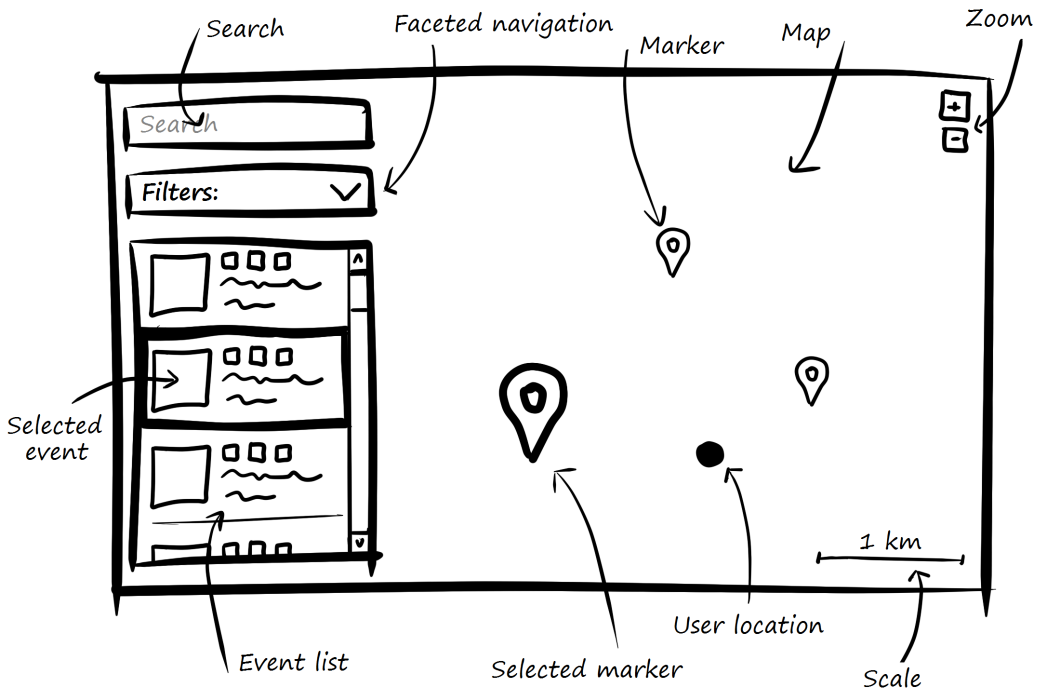


Figure 2.3: Website overview

sidebar. The user can navigate the map in the same way as in Google maps. The map is moved by dragging it with the mouse or by using the search. Zoom level is changed by scrolling in/out or using the zoom in/out buttons.

Search The search bar is located in the top left corner. It is a white bar with *search* written in it. When writing in it a drop-down will appear with location suggestions matching the written text. When the search returns a match the view of the map is moved to center on the location of the match. The event list is updated with the events which are now visible on the map.

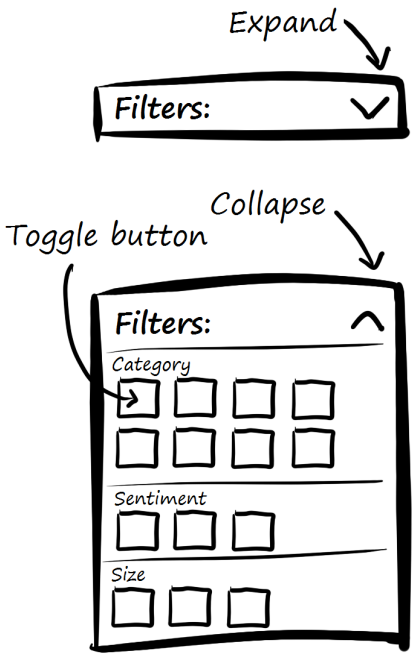
Filters The filtering is located under the search bar. It can be seen in Figure 2.4a. The filtering starts in the collapsed state in order to give more space for the event presentation. It unfolds on click. When unfolded it shows the three filtering possibilities; category, size and sentiment. They are represented as a list of check boxes which are checked as default and can be unchecked to remove that type of events form the list.

Event list The last element in the left side is the event list. This contains a list of events, as seen in Figure 2.3. If no events are available it contains an error message. The list of events can be sorted either by age of the event, distance to the user, alphabetically or by category. The reason for having this list view is to give the user a somewhat detailed overview of the events. The representation of an event in the

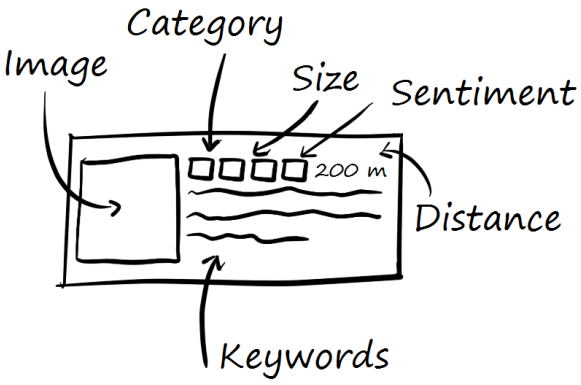
list can be seen in Figure 2.4b. To the left is a picture from the event. Category, size and sentiment are show by descriptive icons to the right of the picture. A short list of keywords summarize the event. The distance to the event is also indicated.

Event description When clicking on an event in the event list or on the map, the view of the event list is changed to a more detailed event description as illustrated in Figure 2.4c. We use the list to represent both the event list and description to limit the amount of elements on the page. This is because we as mentioned want to keep the focus on the map. The event description contains an upscaled version of the picture from the list view alongside other possible pictures that could be from the event. These pictures are arranged in a carousel gallery. The event description also contains additional keywords and a location description. The user can return to the event list view by clicking the back button in the top left corner.

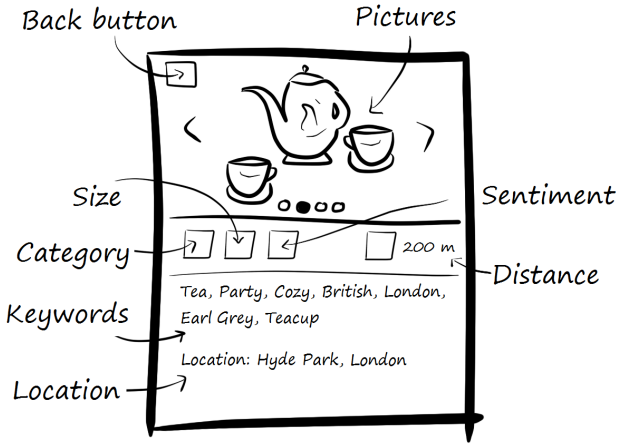
Zoom buttons The zoom buttons are placed at the top right corner with some margin to the sides. It is two buttons one with a plus, for zooming in; and the other with a minus, for zooming out.



(a) Event filter



(b) Event representation in the list



(c) Event description

Figure 2.4: Components in the side bar

3 Background

Throughout our work, we refer to a number of concepts and algorithms, which we describe in the following sections.

3.1 The Bag of Words Model

When we represent data in mathematical form as input for a classification or clustering algorithm, we represent it as a number of numerical or categorical *features*, typically in the form of a *feature vector*. A useful way of representing a text (henceforth referred to as a *document*) as features is to consider a document as a *bag of words*[36]. We discard the order in which terms occur and only look at the frequencies of the individual terms. While this is a great simplification of documents, the useful information is still retained in the representation. We can now represent a document as a high-dimensional vector in which each entry corresponds to a word. The dimension of the vector should be equal to the size of our vocabulary, i.e. the number of unique terms we know of. In practice, however, it is convenient to define a "sufficiently high dimension", as we do not know the exact number of unique terms. To map a term t to an index in a vector of dimension V, we apply a hash function $g(x) : \text{string} \rightarrow \{1, 2, \ldots, V\}$. By using a hash function, we do not need to keep track of the index of each individual term in the word vector. While there is a risk of collisions, the risk is low if we use a good hash function. For the event detection system, we generally use MurmurHash3[4].

3.1.1 Feature Representations

Using the bag of words model, we need to specify how to represent each term with a numerical value. There are several approaches to this, and no representation is superior. In this section, we present three different representations. All representations results in a sparse representation, which is convenient in terms of memory and computational efficiency.

One simple representation is to represent a term t in a document d with a binary value: 1 for present and 0 for absent. This can be useful for small documents such as posts on social media, which usually do not contain several occurrences of the same term. A slightly more elaborate representation is the term frequency $\text{tf}_{t,d}$, which corresponds to the frequency of term t in document d. One drawback of these two methods, however, is that they consider all terms as being equally important. In practice, terms like "the" and "and" do not provide as much information about a document as infrequent words such as "Radiohead". To represent this, we associate each term t with an idf value[36], calculated as shown in Equation 3.1, where df_t is the number of documents containing t and N is the total number of documents. Intuitively, this value increases as the term

gets less common. By multiplying tf and idf, we get the tf-idf score shown in Equation 3.2. This score implicitly handles stop words by assigning them a lower score than less frequent words. Compared to the other representations, this is the only representation that has continuous values. For the event detection system, we use the natural logarithm function and estimate the document frequency on a pre-collected dataset, which we find representative of the general use of English terms on social media.

$$\text{idf}_t = \log \frac{N}{df_t} \tag{3.1}$$

$$\text{tf-idf}_t = \text{tf}_{t,d} \log \frac{N}{df_t} \tag{3.2}$$

3.2 Multinomial Naïve Bayes

The multinomial Naïve Bayes classifier is a probabilistic supervised model. It takes evidence from term frequencies in a document and document collection to infer the probability of a document belonging to a class. According to Bayes Theorem, the probability of a class, $c \in C$ given a document, d, is as shown in Equation 3.3[36].

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} \tag{3.3}$$

Since the normalizer $P(d)$ is a constant among all classes in C, we disregard this term as we are not interested in the normalized probability but just the class with highest probability. The estimated class of a document d, c_d, is calculated as shown in Equation 3.4.

$$c_d = \underset{c \in C}{\text{argmax}}\, P(c|d) = \underset{c \in C}{\text{argmax}}\, P(d|c)P(c) \tag{3.4}$$

To estimate these probabilities, we need to know $P(d|c)$ and $P(c)$. We learn these probabilities from a training set of documents by estimating their frequencies in the samples. The prior probability is estimated as shown in Equation 3.5, where $N(c)$ is the total number of documents in class c. $P(d|c)$ is estimated as shown in Equation 3.6. One important thing to notice is that we assume that the occurrence of a term t is independent of the occurrences of other terms, i.e. the "naïve" assumption. This allows us to calculate the joint probability as a product of probabilities. Each document, d, consisting of n unique terms, is represented as an n-dimensional vector of term counts $[\text{tf}_{t_1,d}, \text{tf}_{t_2,d}, \ldots, \text{tf}_{t_n,d}]$, where $\text{tf}_{t_i,d}$ is the term frequency of term t_i in document d and tf_c is the sum of term frequencies for all terms in class c[36].

$$P(c) = \frac{N(c)}{\sum_{c_i \in C} N(c_i)} \tag{3.5}$$

$$P(d|c) = P([tf_{t_1,d}, tf_{t_2,d}, \ldots, tf_{t_n,d}]|c) \propto \prod_{1 \le i \le n} P(t_i|c)^{\text{tf}_{t_i,d}} = \prod_{1 \le i \le n} \left(\frac{\text{tf}_{t_i,c}}{\sum_{c' \in C} \text{tf}_{c'}} \right)^{\text{tf}_{i,d}} \tag{3.6}$$

When estimating probabilities based on frequencies, an unseen term will result in an estimated probability of zero. In a product of several probability terms, this causes the total probability to be zero, which is inconvenient. Laplace smoothing can handle this problem, by assuming that each term occurs at least once. We now have $P(t|c) = (\text{tf}_{t,c}+1)/((\sum_{c' \in C} \text{tf}_{c'}) + |V|)$, where $|V|$ is the length of our vocabulary, i.e. the number of unique terms we know of[36].

3.3 Logistic Regression

In the naïve Bayes model, we calculate the probability $P(c|d)$ by estimating probabilities $P(d|c)$ and $P(c)$ from our training data. In logistic regression, we learn $P(c|d)$ directly from our training data. Given n features (describing document d) $f_1, f_2, \ldots, f_n$ represented as a vector $\boldsymbol{x_d} = [1, f_1, f_2, \ldots, f_n]^\mathsf{T}$ of dimension $n+1$, we learn a vector of parameters $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_{n+1}]^\mathsf{T}$, such that we have a linear function $\boldsymbol{\theta}^\mathsf{T}\boldsymbol{x_d}$. Because we want the output to be a probability, and hence in the range between 0 and 1, we apply a sigmoid function to the result of the linear function. We can now represent $P(c|d)$ as seen in Equation 3.7, where $P(c|\boldsymbol{x_d}; \boldsymbol{\theta_c})$ denotes the probability of class c given a feature vector $\boldsymbol{x_d}$ (representing the document d) and parameters $\boldsymbol{\theta_c}$[46].

$$P(c|d) = P(c|\boldsymbol{x_d}; \boldsymbol{\theta_c}) = \frac{1}{1 + \mathrm{e}^{-\boldsymbol{\theta_c}^\mathsf{T}\boldsymbol{x_d}}} \tag{3.7}$$

For a binary classifier, we estimate the probability $P(\neg c|d) = 1 - P(c|d)$. To generalize the classifier to predict among $|C|$ classes, $|C| \geq 3$, we train $|C|$ distinct *one-vs.-all* classifiers and predict the class c_d for a document d as the class that has the highest probability, as shown in Equation 3.8. In order to learn the parameters θ, we optimize the mean squared error cost function using the scikit-learn library[41].

$$c_d = \underset{c \in C}{\operatorname{argmax}}\, P(c|d) = \underset{c \in C}{\operatorname{argmax}}\, P(c|\boldsymbol{x_d}; \boldsymbol{\theta_c}) = \underset{c \in C}{\operatorname{argmax}}\, \frac{1}{1 + \mathrm{e}^{-\boldsymbol{\theta_c}^\mathsf{T}\boldsymbol{x_d}}} \tag{3.8}$$

3.4 DBSCAN

DBSCAN[13] is a clustering algorithm which, compared to other algorithms such as k-means, does not need to know the number of clusters. Instead, its parameters are ϵ, which corresponds to the maximum distance between two points in a cluster, and *MinPts*, which defines the minimum number of points within a radius of ϵ of another point before a point is part of a cluster. Since DBSCAN explicitly uses the distance between points, we can use any distance measure. This is convenient for geospatial points, since we can use the distance in meters between geographic coordinates. In addition, by using geographic distance, we need not handle special cases such as the 180$^{\text{th}}$-meridian[1] compared to algorithms using the euclidean distance, and it is easier to pick a sensible ϵ parameter

[1] The meridian where the longitude coordinate goes from -180 to 180

for the model. Compared to k-means, DBSCAN does not make any assumptions about the shape of clusters.

The DBSCAN algorithm differentiates between two kind of points: *Core points* and *border points*. Let the ϵ-neighbourhood of a point p denote the set of points within a distance of ϵ of p. A core point is a point which has at least *MinPts* other points withing its ϵ-neighbourhood. A border point is a point which has less than *MinPts* points in its ϵ-neighbourhood, but at least one core point. All points that are not a core point or a border point are assumed to be noise. DBSCAN is shown in Algorithm 1, which assigns the *CId* attribute of each point to a cluster identifier or *NOISE*. The function NEIGHBOURHOOD(*Points*, p, ϵ) returns the ϵ-neighbourhood of p among all points *Points*.

While the neighbourhood is efficiently calculated using a kd-tree (in the average case), we pre-calculate the pairwise distances between all points and use this to find the neighbourhood of a point in linear time. The main reason for this is simplicity, since we need not keep a data structure updated over time. As long as the number of points is relatively low, we expect this to be reasonably fast.

3.5 Near Duplicate Detection

To measure the similarity of two documents, we consider each document, d, as a set of k-shingles $S(d)$. A k-shingle of a document is a string of k ($k \geq 1$) consecutive tokens in the text. A single token can be a word, a user mention, a hashtag etc. Tokens are converted to lowercase but no further processing is made. The similarity of two documents can be seen as the similarity of two sets of k-shingles. The similarity of two sets A and B can be defined as the Jaccard similarity, which is shown in Equation 3.9. When the Jaccard similarity of two documents is above a certain threshold value, we filter one of the documents. As an example, consider documents d_1 and d_2 with texts "*@jackdd Cheap football tickets at http://www.bit.ly/23g2dg2w*" and "*@peter_sch Cheap football tickets at http://www.bit.ly/23g2dg2w*", respectively. Each document contains three 4-shingles, out of which two are in common ("*cheap football tickets at*" and "*football tickets at http://www.bit.ly/23g2dg2w*"). The Jaccard similarity is $J(S(d_1), S(d_2)) = \frac{2}{3}$. An ideal value of k depends on the length of documents.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.9}$$

3.5.1 Approximate Jaccard Similarity

A naïve approach to near-duplicate detection is to calculate the Jaccard similarity of a new document and all existing documents. However, while Twitter and Instagram posts are usually very short, the large number of documents makes this approach unfeasible. Instead, we use a number of "tricks", described by Leskovec, Rajaraman, and Ullman [35], to approximate Jaccard similarity. First, we hash each shingle into a large sparse vector. We define a hash function $g : \mathit{string} \rightarrow \mathbb{Z}^+$, that maps a shingle to an index in a vector. If a shingle is present in the text, the corresponding entry of the vector has a

Algorithm 1 The DBSCAN algorithm

```
1: procedure DBSCAN(Points, ϵ, MinPts)
2:     ClusterId ← nextId(NOISE)
3:     for all p ∈ Points do
4:         p.CId ← UNCLASSIFIED
5:     end for
6:     for all p ∈ Points do
7:         if p.CId = False then
8:             if EXPANDCLUSTER(Points, p, ClusterId, ϵ, MinPts) = True then
9:                 ClusterId ← nextId(ClusterId)
10:            end if
11:        end if
12:    end for
13: end procedure
14:
15: function EXPANDCLUSTER(Points, p, ClusterId, ϵ, MinPts)
16:    seeds ← NEIGHBOURHOOD(Points, p, ϵ)
17:    if seeds.size < MinPts then                ▷ Too few points in neighbourhood
18:        p.cId ← NOISE
19:        return False
20:    end if
21:    for all s ∈ seeds do
22:        p.cId ← ClusterId
23:    end for
24:    seeds.delete(p)
25:    while seeds.size > 0 do
26:        curP ← seeds.first()
27:        neighbours ← NEIGHBOURHOOD(Points, curP, ϵ)
28:        if neighbours.size ≥ MinPts then
29:            for all n ∈ neighbours do
30:                if n.CId = NOISE ∨ n.CId = UNCLASSIFIED then
31:                    seeds.append(n)    ▷ These are potential core points, so we need to
   visit them again
32:                end if
33:                n.CId ← ClusterId
34:            end for
35:        end if
36:        seeds.delete(p)
37:    end while
38:    return True
39: end function
```

value of 1, otherwise its value is 0. An example of this can be seen in Equation 3.10, where 3-shingles of the text *"mr federer to serve"* are hashed into a 6-dimensional vector. With a reasonable hash function and a sufficiently large vector, the risk of collisions is low. We use the fast hashing function MurmurHash3[4] and a vector of size 2^{32}. By representing shingles as entries in a sparse vector, we save memory and are able to check shingle equality more efficiently than with strings. The problem of comparing sets of shingles has then been reduced to comparing non-zero 32-bit indexes of two vectors.

$$\begin{aligned} g(\text{'mr federer'}) &= 3 \\ g(\text{'federer to'}) &= 1 \\ g(\text{'to serve'}) &= 6 \end{aligned} \quad \rightarrow \quad \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{3.10}$$

For a very large collection of documents, the memory usage with this representation is still rather large, and many pairwise comparisons are still needed in order to find duplicates. To improve performance further, we seek an even smaller representation of the sets. Let x_j denote the index of the first non-zero entry in the hashed vector representation of $S(d_j)$. We now utilize that $J(S(d_1), S(d_2)) = P(x_1 = x_2)$ [7]. We can simply estimate this probability by performing a number, N, of random permutations of the set vectors and count the number of occurrences where $x_1 = x_2$. Instead of actually performing N random permutations, we define N different hash functions, $h_1, h_2, \ldots h_N$. Each hash function, h_i, maps an index in the vector to another index in the vector, i.e. $h_i : \{1, 3, \ldots 2^{32}\} \rightarrow \{1, 2, \ldots 2^{32}\}$. Instead of representing each document d_i as a 2^{32} dimensional sparse vector, we can represent it as a N dimensional *signature* vector, $\text{sig}(d_i)$. Let $\text{sig}(d_i)_j$ denote the jth element in the signature vector of document d_i. Then $\text{sig}(d_i)_j$ contains the index of the first non-zero entry in $h_j(S(d_i))$. Whenever we add a new document d_i to our collection of documents, we calculate and store $\text{sig}(d_i)$. To compare this with a potential duplicate d_j, we generate $\text{sig}(d_j)$ and calculate the approximate Jaccard similarity $\hat{J}$ as shown in Equation 3.11.

$$\hat{J}(d_i, d_j) = \frac{|\{x | 0 < x \leq N \wedge \text{sig}(d_i)_x = \text{sig}(d_j)_x\}|}{N} \tag{3.11}$$

We define the N hash functions as functions of two "basic" hash functions, H_1 and H_2. All hash functions $h_1, h_2, \ldots h_N$ have the form $h_i(x) = H_1(x) + (i-1) \cdot H_2(x)$ mod $2^{32} + 1$. According to [31], this way of generating hash functions works well for randomized algorithms such as this min-hashing. The two basic hash functions we use are MurmurHash3 and xxHash [9], which are both very fast functions.

We can now represent a collection of D documents as an $N \times D$ signature matrix. However, we still need to compare a potential duplicate with all documents in our collection. To avoid this, we use a locality-sensitive hash function to hash documents into a number of buckets, each of which containing documents that are likely to be similar. We split the signature matrix into b *bands*, each containing r rows (i.e. N is divisible by

b). For example, for $b = 2$ and $N = 200$, we split the rows of the signature matrix into bands $1-100$ and $101-200$. We define a hash function $\Pi : (\mathbb{Z}^+)^r \rightarrow \{1, 2, \ldots B\}$ that maps r rows of a signature vector into B buckets. The same hash function is used for all bands, but each band maps to a different array of buckets, i.e. each signature vector is hashed to b different buckets. Only the documents in the b buckets a document is hashed to are potential duplicates, and we only compare a new document with these. For example, consider the signature matrix in Equation 3.12. Each row corresponds to a hash function (in this case we have six), and each column corresponds to a document (1 to 3). We have two bands ($b = 2$) with three rows ($r = 3$), which we use Π to map to two arrays, C_1 and C_2, of $B = 100$ buckets. An example of a resulting mapping could be that document 1, band 1 would be mapped to bucket $C_{1,1}$ and band 2 will be mapped to $C_{2,2}$, document 2 would be mapped to $C_{1,3}$ and $C_{2,4}$, and document 3 would be mapped to $C_{1,5}$ and $C_{2,2}$. Now we need only compare documents within the same bucket. Documents 1 and 3, are compared since these have identical signature vectors in band b_2 and both are mapped to bucket D_2.

$$\begin{matrix} b_1 \\ b_2 \end{matrix} \begin{cases} \begin{pmatrix} 4 & 1 & 2 \\ 1 & 4 & 1 \\ 4 & 2 & 4 \end{pmatrix} \\ \begin{pmatrix} 1 & 3 & 1 \\ 2 & 2 & 2 \\ 1 & 1 & 1 \end{pmatrix} \end{cases} \tag{3.12}$$

3.6 Social Media APIs

Twitter and Instagram provide APIs for accessing their data. This section describes what they provide access to with respect to our needs for event detection.

Twitter

The Twitter API consists of several APIs. The relevant one for us is the streaming API, which notifies a subscriber to the stream whenever new tweets arrive. It is possible to restrict the streaming to only match tweets which contain specific keywords, are tweeted at a specific location, or tweeted by selected users. It seems reasonable to assume that keywords and location are relevant in the context of detecting events. Location is defined by a bounding box. The streaming API provides access to a small percentage of the total volume of tweets being posted every second [55]. We assume it to be one percent of the total amount as detailed in [50] as an upper bound.

Instagram

The Instagram API includes both searching in existing posts and streaming[26]. To use the stream, it is necessary to have a web-server which can be used to receive callbacks

from the API. The callbacks only contain notifications about new posts and not the actual content of the posts[27]. When using the Instagram API, the results can be filtered by location. The location is defined by a center coordinate and a radius which describes an area. The search API does not support keywords, but the results can be restricted to a specific time interval. The Instagram API is limited to 5000 calls per hour [23].

4 Event Detection

The event detection system is a pipeline with several stages. The input to the pipeline is a stream of posts from the social media and the output is a set of events at different time periods. Each stage of the pipeline performs an operation or analysis on the posts. A diagram of the architecture can be seen in Figure 4.1. Each individual stage will be described in more detail in the following sections. From the pipeline's point of view, the posts are fed into the system as soon as they are published on a social medium. As described in Section 3.6, both Instagram and Twitter have API calls to get real-time posts. However, they differ in the methods and post representation. To make a uniform method to access real-time posts we define a simple definition of how posts are represented. This allows us to abstract away the differences between different social media and to use posts from any social medium as long as they can be converted into this format. A post consists of the textual content of a post, the time the post was created, an optional image, and optionally a geographic location from which it was posted as a latitude and longitude pair.

The first part of the pipeline is a number of preprocessing stages. The first two stages are filtering steps. Step one is the language filtering step, described in Section 4.1.1 which removes posts in other languages than English. Step two is the duplicate detection step, described in Section 4.1.2, which removes posts that are near duplicates. This is followed by the location extraction step, described in Section 4.1.3, where we extract place names from posts in order to assign them a location. The posts are collected in a *sliding window* which contains all posts that have been received within the last hour. Whenever more than 10 % of the window are replaced, a snapshot of the data in the window is sent to the next stage of the pipeline. The *Hashtag Clustering* and *Location Clustering* stages, described in Section 4.2, cluster posts based on their hashtags and locations, respectively, in order to group related posts together. *Event Classification* classifies the groups of posts with a binary classifier as events and non-events. The main purpose of this classification is to reduce the number of irrelevant events, e.g. spam, as detailed in Section 4.3. In order to describe the events, we seek to identify some of their contextual properties. In the *Sentiment Analysis* stage, described in Section 4.4, we estimate the overall sentiment of each event. In *Event Categorization*, described in Section 4.5, we assign each event a category. This is followed by *Keyword Extraction*, described in Section 4.6, in which we extract keywords about the event. Finally, in the *Event Location Estimation* stage, which is described in Section 4.7, we estimate the location of the event by looking at the locations of the posts related to it.

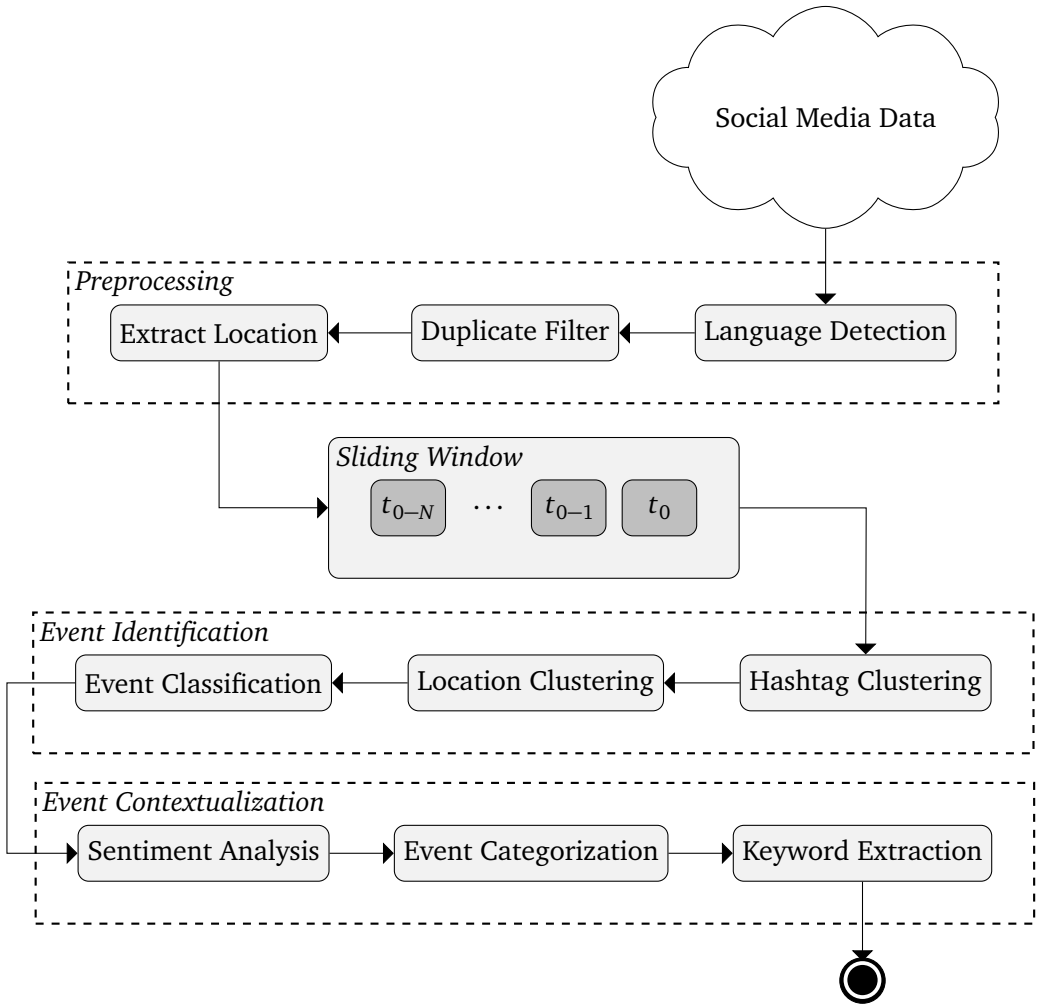


Figure 4.1: Overview of the event detection pipeline. A rectangle is a conceptual stage in the pipeline with data flowing in the order of the arrows. A solid circle with a hollow circle outside represents the output of the pipeline.

4.1 Post Filtering

The first part of preprocessing is to filter out irrelevant posts, and thereby reduce load on later parts of the system. This part contains several stages, as we choose to filter the posts based on two different parameters: the language, which must be English; and the text of the post. We use the post content to remove posts which are near duplicates. We only accept English posts to simplify the later aspects of the data processing.

4.1.1 Language Detection

We are only interested in posts which are written in English. Therefore we filter away posts which are not written in English. To accomplish this we use the langdetect library[48]. The library uses a naïve Bayes classifier to detect the language of the text. The classifier is trained on text extracted from Wikipedia articles and is able to recognize 53 languages, including English.

4.1.2 Near Duplicate Detection

Like most other parts of the World Wide Web, spam takes up a large fraction on the posts on Twitter and Instagram. One example is bots that publish identical posts on several user profiles. To handle this, we detect and filter posts which are near-duplicates of other posts already seen. Posts need not be completely identical to be detected; for example, spam posts with the same message but different mentions of a different user may be detected as well. To efficiently find duplicates in a collection of many posts, we use the hashing approach presented in Section 3.5. Since Twitter and Instagram posts tend to be short, we use shingles of length $k = 2$ and a band size of $b = 20$. To generate the signatures, we perform $N = 200$ random permutations as suggested by [36].

4.1.3 Place Name Extraction

Because a fairly low number of posts are tagged with a location, we extract location information based on keywords in the text. For example, posts can contain words such as "Wembley Stadium", "Big Ben" etc. without explicit coordinates. As a mapping between place names and locations, we use geotagged places from Wikipedia. We have extracted approximately 6 million place names. Other place providers with even more places, such as GeoNames[1], are also available. However, we prefer a small set of well-known places over a large set of small places, as a greater number of places increases the risk of a false-positive location extraction. For example, a naïve approach maps all posts related to "art" to a community in Texas called "Art"[2]. Each place consists of a latitude/longitude pair, a title, and a type e.g. landmark, city, or country.

The posts may contain several keywords which can be interpreted as place names. We apply two heuristics to the place name extraction to reduce this ambiguity:

[1] http://www.geonames.org
[2] https://en.wikipedia.org/wiki/Art%2C_Texas

Place Name Candidates We have observed that it is common to refer to locations in hashtags or by using "@" signs. We therefore only consider hashtags and text that follow an "@" sign as potential place names. We expect that this reduces the number of false-positive detections, since other parts of the text, which tend to be written as natural language, will not be treated as potential locations.

Place Candidate Clustering We notice that common place names tend to be associated with several entries on Wikipedia. In addition, a post may contain several place name candidates. We consider the locations gathered from the place names trustworthy if they are within close proximity of each other. Based on this assumption, we choose to cluster the matching Wikipedia locations using the DBSCAN algorithm. Candidate locations which are further than 1 kilometer from any other candidate location are considered noise. When several locations are geographically close to each other, they form a candidate location cluster. In case of multiple candidate location clusters, we are uncertain about what to do and do not use any of the candidate locations. In case of exactly one cluster, we pick the most specific type of place among all places.

To clarify the procedures, consider the following worked example. We seek to estimate the location of a post with no explicitly associated location. The post contains the text *"Wow what a performance from #louisajohnson #xfactoruk #wembley @ wembleystadium"*, from which we extract the following place candidates: "#louisajohnson", "#xfactoruk", "#wembley", and "wembleystadium". We match these names with place names from Wikipedia and find five matching locations; four matching "wembleystadium" and one matching "wembley". Because we have several place candidates, we apply place clustering which marks the "wembley" result as noise and groups the matches of "wembleystadium" into a single cluster. Since we now have a single cluster, we pick the most specific type of place, which is a "landmark".

Inverted Index

To efficiently lookup a place, we store it in an inverted index. An inverted index is a data store specifically for indexing text. In contrast to a relational database, it stores each token in an index with a mapping to the documents that contains it. While this can be structured in a relational database, an inverted index is usually accompanied of several text-specific utility functions such as tokenizers, stemmers etc. We use Apache Lucene[2] as inverted index. On top of Lucene, we use Apache Solr[3] to provide a REST API for the index.

We configure the inverted index to store the place name in lowercase and with whitespace removed. This allows us to match for example "Wembley Stadium" with "wembleystadium", which is convenient when hashtags are used as place names since hashtags contain no spaces. We also do case folding, which means that apostrophes etc. are removed, since these are seldom used on social media. The same text processing is used when we query for place names.

Together with the place name, we store the coordinates and type of each place, however these values are not indexed.

4.2 Clustering of Posts

Before the posts are classified as events or non-events, we combine them into groups that contain related posts in terms of topics and locations. Typically, posts are shorter than 140 characters, and this amount of information is likely too little for event classification. By grouping posts, we can provide the classifier with more descriptive information about the potential events. In addition, we can remove outlying posts, i.e. posts which do not share topic with any other post or that are geographically isolated from other posts. We group on location and topic based two assumptions. First, we expect that posts from the same event use common hashtags, as described in Section 1.3. Second, we expect that all events have a connection to a physical location, and thereby that all posts in a given area may be a part of the same event.

4.2.1 Hashtag Clustering

This technique assumes that posts which share hashtags are related. We put the posts into one bucket per hashtag, such that all posts in a bucket contain the same single hashtag. One post can appear in several buckets if it contains several hashtags. To discard common hashtags, we only add hashtags with an idf value greater than 8. This value is heuristically defined such that it approximately corresponds to hashtags used less frequently than in $\frac{1}{3000}$ of all posts.

Sometimes, multiple hashtags are used to identify the same event. To handle this case, we assume that some users will use both hashtags in the same post. Based on these posts, we identify buckets describing the same events by combining buckets which have at least $\frac{1}{5}$ of the posts in common. We check for similarity efficiently by using a modification of the near-duplicate detection approach described in Section 3.5. Instead of comparing sets of terms, we compare sets of post ids, corresponding to the buckets. Since posts in a bucket have no particular order, in contrast to terms in a text, we make signatures based on shingle sizes of one. Based on these signatures, we create a distance matrix with the approximate Jaccard similarity between all pairs of buckets and merge buckets with high similarity.

4.2.2 Location Clustering

When we create the sets of posts sharing the same hashtags, we do not take geospatial information into account. One set can for example contain posts from Asia and America at the same time. Since it is not in our interest to discover global events on social media, but to actually assign each event a specific location, we seek to identify different geospatial groups in each set of posts. We do this by performing a DBSCAN clustering within each of the hashtag clusters. We cluster based on the coordinates of each post and the use the geographical distance metric. As parameters for DBSCAN, we set $\epsilon = 1000\,\text{m}$ and

$minPts = 3$. We have heuristically found that these values work well in practice. By clustering on location, we potentially split each topic into several geographically isolated events.

4.2.3 Duplication Removal

After having performed hashtag and location clustering, we have identified cases of events occurring twice within a short geographical distance. These occurrences are likely due to difficulties in picking a good similarity threshold for hashtag clustering. Since we are unable to pick a value for the similarity threshold that works well in practice, we fix the problem heuristically by applying a kind of hierarchical clustering. This can be seen as a clustering of the clusters found by location clustering. In this clustering, we again use DBSCAN though with the average coordinates of each cluster from location clustering as samples. We use an ϵ value of 1000 m and a *minPts* parameter of 1. This means that no points will be presumed to be noise; isolated clusters will stay unmodified. Within each of the resulting clusters, we calculate the exact Jaccard similarity between the set of hashtags present in location clusters. If the set of hashtags in two location clusters have a Jaccard similarity of at least 0.3, we assume that they describe the same event and merge them.

4.3 Event Classification

Having separated social medium posts into different groups, we seek to distinguish those representing events from those that do not. In order to do this, we create a classifier to which the input is a list of posts and the output is an assignment to a binary class: event or non-event.

4.3.1 Training Data

To train the classifier, we need a dataset of labeled posts from social media. Collecting this data can be very time consuming, since many samples are needed. Because of time constraints, we choose make a naïve assumption which allows us to automate the process. The assumption is that all posts containing a hashtag related to an event is actually about the event. We now collect positive samples by collecting all posts containing event-specific hashtags. Similarly, we collect negative samples by collecting posts with hashtags which we expect are not related to events.

A potential problem with this approach is that we risk overfitting the classifier against the specific events that we have collected. As a result, it will not generalize well over unseen samples. As an attempt to avoid this, we conservatively remove all hashtags we have collected with from the posts. The collected dataset contains 125 171 labeled posts, out of which 117 799 are related to events and 7372 are not.

4.3.2 Feature Representation

As input for the classifier, we only look at the text of posts. While other parts, such as the number of posts over time, may be useful, we do not have labeled data to train such a model. We adjust the text in tree different ways, related to representation of hashtags.

Without Hashtags We adjust the text by removing all the hashtags, and thereby only look at the normal text.

Without the "#" Symbol We remove the "#" symbol from the text, so hashtags look like the normal text.

With Hashtags We keep all the hashtags in the text.

To represent the text, we tokenize and lowercase it and hash each token into a high dimensional vector as described in Section 3.1.

4.4 Sentiment Analysis

To present events in a descriptive way for users, we estimate the overall sentiment of an event. The sentiment can help users get a feeling of the general mood of an event and as a result make it easier for them to choose events suited for their needs and moods. We train two classifiers to judge the sentiment of events. We train a naïve Bayesian classifier and logistic regression classifier. The theory behind these methods are described in Section 3.2 and Section 3.3.

4.4.1 Training Data

Instead of collecting training data ourselves, we use a large collection of sentiment-labeled tweets collected by [18]. The authors have automatically labeled the data as positive or negative based on emoticons present in the text [19]. According to [1], the classifier trained by [18] is the classifier with the highest accuracy where the training data is publicly available.

4.4.2 Feature Representation

For sentiment analysis, we present several different feature representations based on the bag-of-words model. In all representations, we hash terms into a high-dimensional vector. For the specific values in the vector, we use either binary values, tf values, or tf-idf values.

One problem with the bag-of-words model for sentiment analysis is that the order of words can be very important. One example is negation in phrases such as *"I don't like this event"*. A naïve bag-of-words model would not be able to differentiate the phrase sentence from *"I like this event, don't you?"*. To handle the case of negation, we use a method presented in [40] and [10], which is to add the "_NEG" suffix to all terms following a negation word. For example, the previous phrase is converted into *"I don't*

like_NEG this_NEG event_NEG", because "don't" is a negation word. We compare negation handling with no negation handling. This is in total six different feature representations. As suggested by [42], we do not perform stemming or stop word removal on the text, as this can potentially remove valuable terms such as "not".

The user interface is designed to show the overall sentiment of an event rather than for example the fraction of positive and negative posts. Because of this, we do not perform sentiment analysis on individual posts but instead concatenate the text of each post in an event into a single entity and then classify that entity.

4.5 Category Classification

To identify the category of an event, we train a classifier to assign it a category. We only assign a single category to each event. The categories we assign are *business, entertainment/arts, politics, science, sports, tech, and world*. While other more meaningful categories could be defined, we choose these categories on the basis of the availability of training data.

4.5.1 Training Data

The training data for the classifier is a collection of labeled news articles collected by [29]. While news articles and tweets are two different kinds of media, we assume that, given a sensible feature representation, the classifier can generalize from news to any kind of text. The news collection has been collected by manually selecting a set of category-specific RSS providers. All news articles from a single provider is assumed to be of a specific category. Each category consists of approximately 6000 articles.

4.5.2 Feature Representation

We solely use features constructed from the post text to classify event categories. As previously, we tokenize and lowercase the text of each post and hash each token into a high dimensional vector. We evaluate feature representations with binary values, tf values and tf-idf values.

4.6 Keyword Extraction

To present an event in a meaningful way, we extract describing and important keywords from the posts. To simplify this procedure, we assume that hashtags in event-related posts are used to describe the event and that they can be used directly as keywords. In addition, we assume the most relevant hashtags have a high tf-idf value among all existing hashtags. To find the keywords for an event, we therefore look through all the hashtags in posts about the event. We calculate the tf-idf value for each hashtag and use the 10 hashtags with the highest tf-idf values.

4.7 Event Location Estimation

Many of the posts which represent an event contain coordinates. To find the location of an event, we calculate the *center of gravity* for the posts with coordinates, using the geographic midpoint method[38]. The method assumes a spherical earth, but when the posts are close, which we make sure they are by performing location clustering, this assumption provides sufficient accuracy. The resulting point is considered the location of the event.

5 Experiment

In this chapter, we present the experiment and discuss the event detection system and the event contextualization. We evaluate to which degree parts of the presented functional requirements from Section 2.1 are fulfilled. We first describe the experimental setting, which includes the evaluation metrics and datasets we use for the evaluation. Following are the results for individual parts of the event detection pipeline and for the overall event detection system.

5.1 Experimental Setup

In the following sections, we present the setup for the evaluations we perform, which includes the datasets and metrics used throughout the evaluation.

5.1.1 Datasets

Throughout the evaluation, we use several different datasets of social medium posts. We refer to each dataset using the term denoted in parenthesis, e.g. DS1.

Edfringe Festival (DS1)

The Edfringe dataset has been collected by the *ForgetIT Project* [17] and contains 30 388 Twitter posts collected during the Edinburgh Festival Fringe in 2014. To make the dataset more useful for our work, we add posts collected from Instagram published during the same period with the "#edfringe" hashtag (677 additional posts). In total, this dataset contains 31 065 posts, out of which 21 981 contains coordinates. This distribution of posts containing coordinates does not represent the general distribution on Twitter and Instagram.

Twitter Language Dataset (DS2)

The Twitter language dataset is provided by Twitter[53] and contains tweets which have been manually labeled with their corresponding languages. The language distribution in the dataset is equal to that of all tweets. The sample contains 120 575 tweets out of which 31 % are English.

Arbitrarily Sampled Data (DS3)

During the preparation of event detection, we have collected data from Twitter and Instagram during arbitrary (non-consecutive) time intervals. This dataset contains 381 340 posts, out of which 160 040 has associated coordinates. The dataset contains 323 878 posts from Twitter and 57 462 posts from Instagram.

An Hour in London (DS4)

This dataset has been collected in London on November 17 from 20.00 to 21.00 UTC. The dataset contains 9131 posts, out of which 2778 contain coordinates. 7096 posts are from Twitter and 2778 are from Instagram. For each post, we have labeled them as events or non-events. If a post is part of an event, we have annotated it with a short description and the location of the event. This dataset is useful for event detection evaluation, as it is labeled and collected in an English speaking city during night where most events happen. In total, 53 distinct events are mentioned in this dataset.

The dataset should ideally be labeled by several people to ensure the quality of the labeling and to compensate for human errors. However, due to time constraints, each post has only been labeled by a single person.

Event Hashtag Posts (DS5)

This dataset is detailed Section 4.3. It contains automatically labeled posts of events and non-events.

Sentiment140 Data (DS6)

This dataset is presented in Chapter 4.4 and contains Twitter posts labeled as negative or positive.

Categorized News (DS7)

The categorized news dataset is described in detail in Section 4.5. It contains news articles labeled with category.

5.1.2 Classifiers

For classifier evaluations, we evaluate all classifiers using both a naïve Bayes classifier and a logistic regression. For the naïve Bayes classifier, we only use the discrete feature representations, i.e. binary and tf.

5.1.3 Evaluation Metrics

To evaluate classifiers, we use precision, recall, and F_1 scores as metrics. These three scores together summarize the quality of the classifiers. Precision and recall are calculated as shown in Equation 5.1a and Equation 5.1b, respectively[36], where true positives (tp) is number of correct positive predictions, false positives (fp) is the number of false positive predictions, and false negatives (fn) is the number false negative predictions. The F_1 score, as seen in Equation 5.1c, is a combination of both precision and recall. As mentioned in Section 3.3, we use several binary classifiers when classifying one of many

classes. In this case, the evaluation metrics are weighted averages of the scores with respect to the number of samples for each class.

$$Precision = \frac{tp}{tp + fp} \tag{5.1a}$$

$$Recall = \frac{tp}{tp + fn} \tag{5.1b}$$

$$F_1 = 2\frac{Precision \cdot Recall}{Precision + Recall} \tag{5.1c}$$

To evaluate clustering methods we use normalized mutual information (NMI). NMI is a measure for how much information about a post we gather by knowing which cluster it belongs to. NMI is a number between 0 and 1. When the NMI is zero, knowing which cluster a post belongs to give us no new information about a post, i.e. a random clustering of posts. When the NMI is 1, knowing the cluster which a post belongs to gives us perfect information about the class of the post, i.e. a perfect clustering of the posts[36]. In our case, the class of a post is a combination of the topic and area of the post, which may or may not be an event. According to [36] the mutual information of a cluster is formally defined as shown in Equation 5.2, where $\Omega = \{\omega_1, \omega_2, \ldots, \omega_K\}$ is the set of clusters and $\mathbb{C} = \{c_1, c_2, \ldots, c_J\}$ is the set of classes. ω_k is the set of posts in ω_k and c_j is the set of posts in c_j. $P(\omega_k)$ is the probability of a posts being in cluster ω_k, $P(c_j)$ is the probability of a post being in class c_j, and $P(\omega_k \cap c_j)$ is the probability of a post being in the intersection of ω_k and c_j. These probabilities are estimated from the actual result of the clustering.

$$I(\Omega; \mathbb{C}) = \sum_k \sum_j P(\omega_k \cap c_j) \log \frac{P(\omega_k \cap c_j)}{P(\omega_k)P(c_j)} \tag{5.2}$$

We use the Scikit-learn[41] implementation of NMI which uses $\sqrt{H(\Omega)H(\mathbb{C})}$ as normalization factor, where $H(\Omega)$ and $H(\mathbb{C})$ is the entropy of the clustering and the classes, respectively. Entropy is defined as $H(S) = -\sum_{s \in S} P(s) \log P(s)$. Putting all that together we reach the following definition of NMI:

$$\text{NMI}(\Omega, \mathbb{C}) = \frac{I(\Omega; \mathbb{C})}{\sqrt{H(\Omega)H(\mathbb{C})}} \tag{5.3}$$

5.2 Post Filtering

In this section, we evaluate the preprocessing stages in the event detection pipeline individually and in isolation of the other stages of the pipeline.

5.2.1 Language Detection

In this section, we evaluate whether the language detection library can generalize from news articles to Twitter posts and test its efficiency on social media posts.

Precision	0.9659
Recall	0.8532
F_1-Score	0.9061

Table 5.1: Results of language detection on the DS2 dataset

Effectiveness

According to the author of the library we use, the library has a precision of more than 99 % for language detection among all languages, and a precision of 100 % for detecting English [49]. However, the language detection is evaluated on news articles, and it is not given that it works equally well for social media posts, which are usually informal and very short. To ensure the method works sufficiently well for our use case, we evaluate the language detection on social media posts. As we only want to detect if a post is English or not, we limit the evaluation to how well the library recognizes English. We evaluate on the DS2 dataset.

We evaluate the language detection by predicting language of tweets with the language detection library and compare the predicted results with the labels. The results can be seen in Table 5.1. The precision is not as good on tweets as it is on news articles. However, for our use case, a precision of 0.9659 is sufficient as the number of false-positive posts is likely too small to influence the event detection. The recall is 0.8532, which is more problematic, as we will likely miss English posts that could potentially be useful. However, we expect that significant events have sufficiently many mentions to still be represented in the set of posts used for event detection. The results could potentially be improved by looking at the estimated probabilities for the various languages and include posts that the classifier is uncertain about.

Efficiency

We evaluate the performance by selecting a random sample of 1 million texts from the DS1 dataset. We assume that the length of posts from this sample corresponds to the global distribution of post length. The results are shown in Figure 5.1. The spike at $x = 0$ is caused by the fact that the classifier loads the data first time it runs. As can be seen, language detection takes constant time as a function of the number of documents. The performance is sufficiently fast and scales very well to large datasets.

5.2.2 Near Duplicate Detection

We evaluate the near duplicate detection method on effectiveness, since the method is an approximation, and on efficiency, as we seek to perform the detection fast.

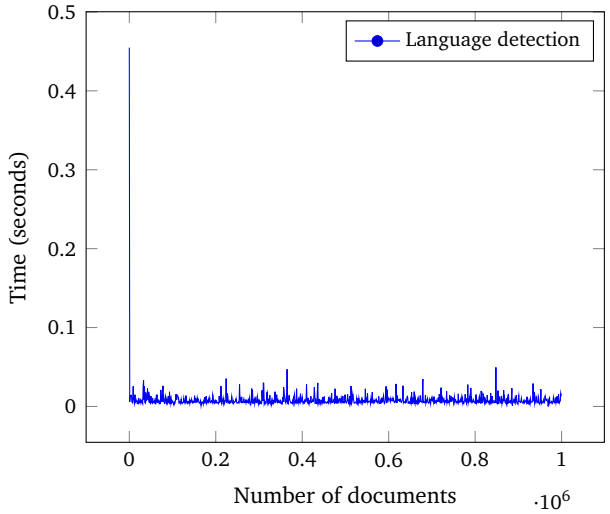


Figure 5.1: Efficiency evaluation of language detection on the DS1 dataset

Effectiveness

To evaluate the accuracy of the near duplicate detector, we compare the approximated Jaccard similarity to the true Jaccard similarity. We compare 315 posts from the DS3 dataset. For each post, we compute the Jaccard similarity between every other post using both the approximation approach and the true Jaccard similarity (in total 99 225 comparisons. If Jaccard similarity is greater than 0.9, we mark it as a duplicate, and otherwise as a non-duplicate.

The results show that the approximation approach and the true Jaccard similarity agree on 328 comparisons as duplicates and 98 895 as non-duplicates. There are two disagreements in total. In both cases, our approximation approach find a false-positive duplicate. While the variance is likely high due to the low number of evaluation samples, the results are good and we conclude that our approximation approach is sufficiently accurate.

Efficiency

To evaluate the efficiency of the near-duplicate detection, we use the DS1 dataset, as we expect this to represent a realistic distribution of similar posts. The result of the evaluation can be seen in Figure 5.2. The performance scales linearly as a function of documents. As of 2013, the average number of tweets per second is 5700[33]. Given this rate of posts, we have (without parallelization) approximately 175 µs per post. Since we only work with a small subset of the data, this is unlikely to be a problem. However, in case we get access to more data, we will need to handle this differently. For example, we could relax the requirements of duplicate detection such that it finds exact duplicates. By doing that, we can represent the text in each tweet by a single hash code and need not consider overlapping shingles.

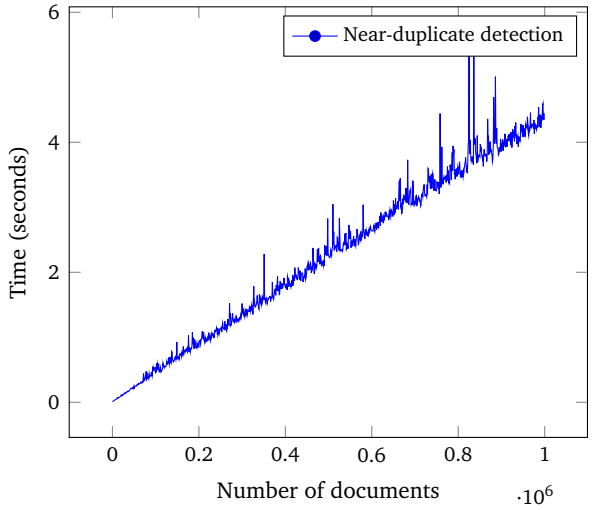


Figure 5.2: Efficiency evaluation of near-duplicate detection on the DS1 dataset

5.2.3 Place Name Extraction

To evaluate the precision of place name extraction, we extract locations from the posts in the DS3 dataset that contain coordinates. For each post, we estimate the location based on place candidates, if any, and calculate the distance (error) between the estimated location and the actual location. We evaluate both the naïve approach with place type prioritization, where we assign the best matching location in terms of text similarity, and the clustering approach previously described. Out of all posts, the place extraction assigns 25 424 a location; the rest are skipped due to uncertainties or missing place candidates. The results of both methods are shown in Figure 5.3. The median error of the naïve approach is 2809 m whereas for the clustering approach it is 3141 m. From median error and the histogram, it can be concluded that, while the two methods perform similar, the naïve approach works slightly better than the clustering approach. Both errors show that most place extractions are several kilometers from the actual location though. However, a great fraction of the posts are within 1000 meters of the actual location.

The reasons for the relatively large errors may be that people write posts only including hashtags of the city they are located in rather than a more specific place. A city covers a very large area, yet it is represented as a single coordinate pair in the place extraction index. Another potential cause of error is that people watching e.g. a national football game may write the name of both participating countries. It is possible that the results would be better without extracting locations from large-area places such as countries and cities. Another possible issue is that we extract the wrong location, as there are several places with the same name. However, since the alternative to doing place extraction is to have no location at all, we expect that place extraction does contribute with useful information despite its inaccuracy.

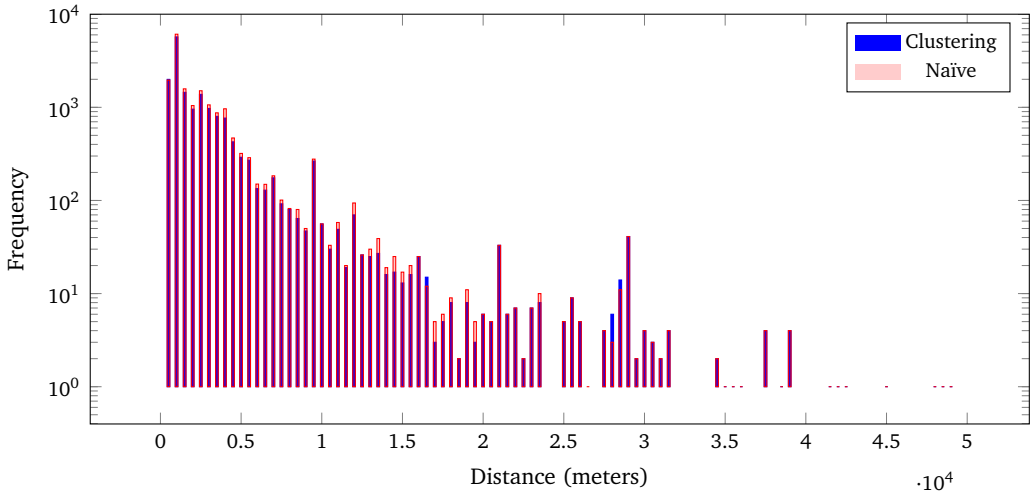


Figure 5.3: Distribution of place name extraction errors on the DS3. Note that the y-axis is logarithmic.

5.3 Clustering of Posts

Due to the high coupling between the clustering stages, we evaluate all three stages together.

5.3.1 Evaluation Method

We use two metrics to evaluate the clustering step: Normalized Mutual Information (NMI) and recall. We calculate the recall of the clustering by categorizing each cluster as representing an event. The most represented event in a cluster determines which event the cluster represents. As the DBSCAN algorithm may filter some posts, some events may end up not being represented in any cluster. Also as a cluster can only represent one event, some clusters may be so large that they contain posts from several events. In this case the small events are swallowed by the large event in that cluster. We use the DS4 dataset as input to the clustering step.

5.3.2 Results

For the clustering algorithm we get a NMI score of 0.4765 and a recall of 4/53 (0.0755). An NMI of 0 represents a randomized clustering, and a NMI of 1 is the perfect clustering. Our clustering lies roughly midway between these two extremes, so while the clustering is not worthless, there is still very much room for improvement. The hashtag clustering parameters is set to filter events with less than five posts. When only considering events with five or more posts the recall is 4/12 (0.3333), which is significantly better. Still many small events are “swallowed” because they are put into the same cluster. We

suspect this is because the idf values for some common hashtags are unrealistically high, leading to large clusters which are too general.

The clustering may be improved by decreasing the minimal cluster size from five to two or three. While we only find four clusters which are events, they have to be quite pure for them to be recognized as representing an event at this stage. Further on in the pipeline, some of the clusters which mostly contain noise may still hold enough information for us to recognize events within them, even though we categorize them as noise clusters in these results. The assumption that posts can be clustered by hashtags seems to be valid, as the posts are clustered reasonably well, considering the issues with the idf values.

5.4 Event Classification

The classification of events is evaluated with both different feature representations and different classifiers. We test both the naïve Bayes classifier and the logistic regression with text representations as binary, tf, and tf-idf, values. In addition, we evaluate three text modifications: without hashtags (H1), without "#" (H2), and with hashtags (H3). We first evaluate the classification on individual posts in the DS5 dataset using 10-fold cross validation. However, since we in practice will use the classifier on concatenated text from several posts about an event, we also evaluate it on the data in DS4. As the former event dataset is skewed, the multinomial naïve Bayes model is evaluated both with and without a prior. However, this factor turns out to have no significant implication for the results and is therefore not included in the rest of the evaluation.

The results of evaluation can be seen in Table 5.2. All scores are generally very high on the individual posts. However, due to the skewed DS5 dataset, a classifier that always predicts *event* will predict 94 % of the posts correct. The high scores can also indicate that the models may have overfitted the training data. While we have used cross-validation to avoid overfitting individual posts, it does not ensure that we do not overfit in a way such that the models learn to detect the specific events we use for training rather than events in general. It is uncertain whether we have enough data to learn a model to generalize.

To conclude whether we have overfitted the data, we also evaluate on the DS4 dataset, in which all event and non-event samples are distinct from those that we use for training. Since the dataset contains labels for individual tweets, we do not have several distinct non-event groups. To compensate for this, we generate 20 non-event clusters, each of which from 12 random non-event posts. As it can be seen in the results two different combinations of features and models are equally good in classifying events on the manually labeled data, namely logistic regression with tf-idf and no hashtags (H1), and multinomial naïve Bayes with tf and normal hashtags (H3). We can therefore chose to use either of them. Since both are just slightly better than a random classification, this indicates that the model did indeed overfit the training data. To further conclude on this, the overall event detection pipeline should be evaluated both with and without the event classifier enabled.

			DS5			DS4		
Model	Fea.	"#"	Precision	Recall	F_1	Precision	Recall	F1
NB	tf	H1	0.9469	0.9465	0.9250	0.4444	0.6667	0.5333
NB	tf	H2	0.9490	0.9484	0.9291	0.4286	0.7500	0.5455
NB	tf	H3	0.9487	0.9482	0.9286	0.5000	0.7500	0.6000
LR	tf	H1	0.9902	0.9868	0.9877	0.3810	0.6667	0.4848
LR	tf	H2	0.9923	0.9892	0.9901	0.3462	0.7500	0.4737
LR	tf	H3	0.9917	0.9885	0.9894	0.4286	0.7500	0.5455
LR	tf-idf	H1	0.9844	0.9816	0.9820	0.5000	0.7500	0.6000
LR	tf-idf	H2	0.9852	0.9824	0.9829	0.4091	0.7500	0.5294
LR	tf-idf	H3	0.9852	0.9824	0.9829	0.4286	0.7500	0.5455

Table 5.2: Results of the event classification experiment with the DS4 and DS5 datasets

5.5 Overall Event Detection

While most individual stages perform well in isolation, new problems may arise when combining them. This section evaluates the system with respect to its overall purpose: to detect events from social medium posts. We evaluate the system with the following metrics:

Precision and Recall of Event Detection It is important to have both high recall and precision to ensure that users are presented with a variety of events, yet avoid filling the list with spam, i.e. non-events.

Actuality Events must be detected sufficiently fast to ensure that they are presented to the users before the events are finished.

Precision of Event Location The reported location must be nearby the actual event, as a user should be able to locate the event.

5.5.1 Evaluation Method

To measure precision and recall for the system, we use the manually labeled DS4 dataset as the "ground truth". We use the dataset without labels as input for the system, and compare the resulting events with the labeled data. We choose to measure two different recall scores; one based on all events there exist in the manually labeled data, and one with only events containing at least five posts. Events with less than five post contain too little information for the system to detect them.

Event actuality is measured by manually finding the time periods during which the detected events happen. Location accuracy is measured as the distance in meters from the actual location to the detected location. In case an event covers a large area, any location within this area is estimated to have a distance of zero meters.

5.5.2 Precision and Recall

The precision and recall results obtained by running the system on the DS4 dataset are listed in Table 5.3. As the performance of the event classifier turned out to be poor, we evaluate the performance of the system with and without the event classification step. During the clustering step of the system, topic clusters containing less than five posts are removed. Only 12 of 53 events in the data fulfills this requirement. As can be seen in Figure 5.4, most events contain less than five posts, which are indicated by a red color, and hence provide very little information. We therefore provide two values for every metric: One computed from only large events (*Large*), i.e. events with five or more posts, and one for all events (*All*).

	Without Classifier		With Classifier	
	Large	All	Large	All
Precision	0.1714	0.1714	0.2000	0.2000
Recall	0.5000	0.1132	0.3333	0.0755
F_1 Score	0.2553	0.1363	0.2500	0.1095

Table 5.3: Results of overall event detection on the DS4 dataset

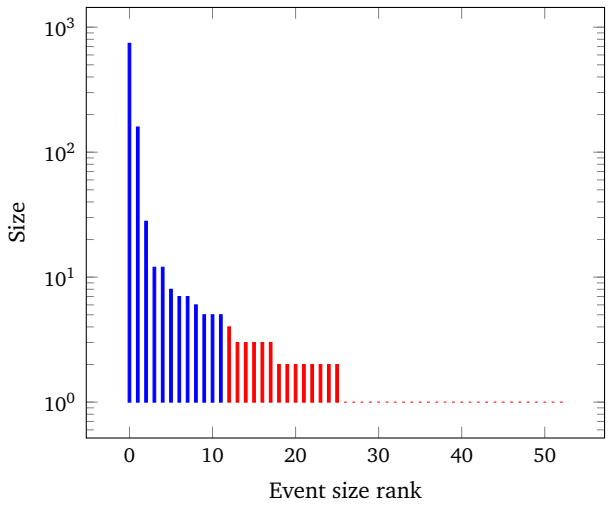


Figure 5.4: Size of the individual events in the DS4 dataset ordered from largest to smallest event. Note that the y-axis is logarithmic.

The event classifier does improve the precision of the results (by 15 %), but the recall also goes significantly down (by 40 %). Even though we value precision higher than recall, the penalty is too great in this case for the classifier to be useful, which is also reflected by the lower F_1 score.

There are several areas in which the precision and recall of the system can be improved.

Some events are overlooked during hashtag clustering, as small events can be swallowed by a large cluster of irrelevant posts. Other events are overlooked even though there are several posts from that event because the posts do not contain common hashtags. Another problem is that we have many false-positive events. We suspect that this may be caused by unrepresentative idf values for the hashtag clustering, causing normal hashtags to be considered as potential events. We estimate that our functional requirement of finding current events from social media is partially fulfilled.

5.5.3 Event Actuality

All events detected by the system were ongoing at the time they were detected. While there was a small number of mentions of historical or future events in the evaluation data, these were not detected as events in any of the evaluated setups. The reason for this is likely that ongoing events get more attention on social media than historical and future events, which is an important parameter in our event detection system.

5.5.4 Precision of Event Location

We calculate the error distances between the actual location of detected events and the location estimated by the system. The average error for events detected without using the event classifier is 95 m, and the average error for events detected with the event classifier enabled is 118.75 m. The largest error is 200 m, which indicates that our concerns regarding imprecise place name extraction have little influence on the final result. We estimate that the precision is sufficiently accurate for people to locate the events given their predicted location. This fulfills our functional requirement of finding the locations for events.

5.6 Event Contextualization

Because the event contextualization part is primarily used for the user interface, we evaluate this separately from the actual event detection system. We first evaluate the sentiment analysis, followed by the event category classifier, and finally the keyword extraction method.

5.6.1 Sentiment Analysis

Due to time constraints, we evaluate the sentiment classifier on a random sample of 100 000 posts from the DS6 dataset. We evaluate the different feature representations and classifiers using 10-fold cross validation. The results can be seen in Table 5.4. As can be seen, the logistic regression model generally performs best on all feature representations. The binary feature representation with negation handling gives the highest score for both models, so we use the logistic regression model with binary features and negation handling. This accomplishes our functional requirement of finding the sentiments of events.

Model	Feature Entries	Negation	Precision	Recall	F_1
Naïve Bayes	binary	no	0.7619	0.7557	0.7541
Naïve Bayes	binary	yes	0.7661	0.7655	0.7654
Naïve Bayes	tf	no	0.7337	0.7312	0.7314
Naïve Bayes	tf	yes	0.7408	0.7407	0.7401
Logistic regression	binary	no	0.7794	0.7792	0.7791
Logistic regression	binary	yes	0.7828	0.7826	0.7826
Logistic regression	tf	no	0.7495	0.7495	0.7495
Logistic regression	tf	yes	0.7522	0.7522	0.7522
Logistic regression	tf-idf	no	0.7562	0.7561	0.7561
Logistic regression	tf-idf	yes	0.7520	0.7520	0.7520

Table 5.4: Sentiment analysis results on a subset of the DS6 dataset

Model	Feature Entries	Precision	Recall	F_1
Naïve Bayes	binary	0.6256	0.5667	0.5031
Naïve Bayes	tf	0.6234	0.5461	0.4845
Logistic regression	binary	0.7864	0.7908	0.7869
Logistic regression	tf	0.7693	0.7740	0.7689
Logistic regression	tf-idf	0.7482	0.7534	0.7486

Table 5.5: Category classification results on the DS7 dataset

5.6.2 Category Classification

We evaluate the category classification in two ways. First, we evaluate it on the labeled news articles (DS7) using 10-fold cross validation, and afterwards we evaluate the best classifier on events from the dataset collected during an hour in London with manually categorized events.

The results made using the DS7 dataset can be seen in Table 5.5. The logistic regression classifier has an F_1-score of 0.7869 while the best representation for the naïve Bayes classifier has an F_1-score of 0.5031. The best feature representation for logistic regression is binary values.

To make sure the logistic regression classifier with binary features can generalize from news to events, we evaluate it on the DS4 dataset. The result of this is a precision-score of 0.7589, a recall-score of 0.8302, and an F_1-score of 0.7879. This shows that the classifier performs only slightly worse when generalizing. Hence, we conclude that the classifier is useful for presenting an event to the user, which fulfills our requirement of finding a category for a event.

5.6.3 Keyword Extraction

We evaluate the keyword extraction method by assessing the relevance of each keyword relative to the event they describe. Some keywords do not contribute with additional information about an event. For example, given the keyword "Djokovic", another keyword "NovakDjokovic" does not contribute with new information, as they both refer to the same person. Because of this, we evaluate both the number of total relevant keywords and the number of unique relevant keywords, i.e. the number of keywords that contribute with unique information.

The keywords extraction is evaluated on the events correctly detected by the system on the DS4 dataset. The results can be seen in Table 5.6. Since detected events have very different sizes, and this may influence the quality of the extracted keywords, the size of the events are listed as well.

	Relevant keywords	Unique keywords	Event size
Event 1	5/10	4/10	25
Event 2	4/10	3/10	6
Event 3	6/10	3/10	7
Event 4	4/10	3/10	12
Event 5	9/10	6/10	743
Event 6	9/10	5/10	159
Average	6.2/10	4/10	

Table 5.6: Results of the keyword extraction experiment on the events detected by the system with the DS4 dataset

On average, approximately half of the extracted keywords are relevant for the event. Only few useful keywords about an event are provided, and this is not sufficient for representing the event a user. The other half of the keywords is noise that may disturb the understanding of the event. This makes it difficult for a potential user to get an idea of what the event actually covers.

The quality of keyword extraction largely depends on the size of the event. In the large events, almost all keywords are relevant, though many are redundant. The more posts about an event, the greater variety there is in the keywords used to describe it. Adding a threshold value for determining when a keyword is likely to be relevant for an event may remove some of the noise. The erratic quality keyword extraction makes our requirement for finding keywords of events only partially fulfilled.

6 Front End Implementation

In this chapter we describe the construction of the front end. We start by explaining the architecture of the web application, and which platforms it supports. Next we describe the user interface design and the differences from the design proposed in Section 2.3. Lastly, we present a usability evaluation of the interface.

6.1 Web Application Architecture

The web application is a single-page application. Client side scripting dynamically updates the web page in response to user input. The client retrieves information by sending requests to an API which provides access to our event database. The structure of the web application is illustrated in Figure 6.1.

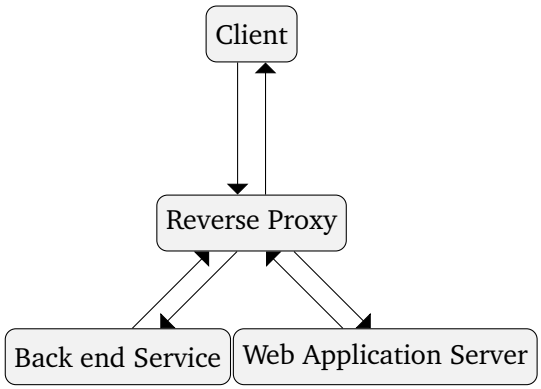


Figure 6.1: Web application achitecture

Client Module The client module generates the web page, client side. JavaScript is used to generate the HTML for the page. We use the Bootstrap client side framework for the web application.

Reverse Proxy A reverse proxy is used to route the traffic between the client and the servers. We use nginx[1] as our reverse proxy server. We use a remote proxy server to hide the fact that the client communicates with several different servers. The requests are routed to different servers depending which resource is requested. If the resource identifier begins with `'/api'` it is routed to the back end service, otherwise it is routed to the web application server. The remote proxy can also be used as a load balancer if several computers are able to service the request.

[1] `http://nginx.org/`

Web Application Server The web application server hosts the HTML and the code that is needed to render the web page. The web application server uses the Flask[2] web application framework. We use the templating language provided by Flask to dynamically create parts of the web page. Thus the page is partially generated server side, and partially generated client side. We use Gunicorn[3] as the actual web server software to run the Flask web application.

Back End Service The back end provides access to the ongoing events which are stored in a database. The server implements a RESTful API which the client uses to retrieve data about ongoing events and is also powered by Gunicorn and Flask. The API has two resources `'/api/events'` and `'/api/events/all'`. They only accept GET requests as the API is only meant to be used for data access. The resource `'/api/events/all'` returns all events in the database and `'/api/events'` returns all events within a geographic bounding rectangle. The rectangle is defined as two coordinates formatted as four parameters, `'sw_lat'`, `'sw_lon'`, `'ne_lat'`, and `'ne_lon'`, which must be supplied. The responses from both resources consist of a JSON formatted list of event descriptions.

We make use of the MVC-WEB (Model-View-Controller web) pattern[20]. MVC-WEB is an architectural model which separates an application into three parts, as seen in Figure 6.2. The three parts are the model, the view, and the controller. The view is the code and markup responsible for presenting information to the user. The controller provides information to the view, and the view triggers events in the controller when the user interacts with the view. When the controller receives an event which requires it to manipulate data, the controller sends requests to the model. The model contains the data of the application and the business logic which manipulates the data. This separation keeps the coupling between the view, the controller, and the model loose, and the cohesion of the individual modules high. Our design differs from the MVC-WEB pattern in that MVC-WEB allows the view to query the model directly, but we avoid direct coupling between the view and the model.

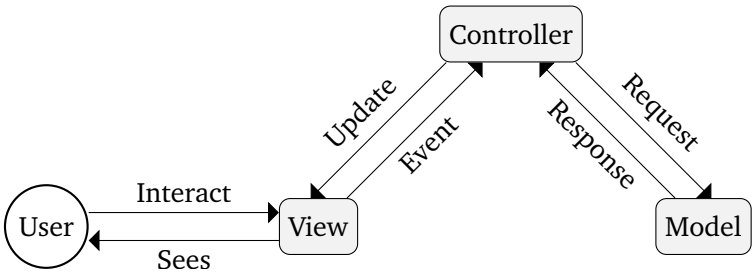


Figure 6.2: Model-View-Controller Pattern. The circle represents an actor and rectangles represent the conceptual components that make up the MVC pattern. Arrows represent the direction of communication between components.

[2] http://flask.pocoo.org/
[3] http://gunicorn.org/

6.2 Supported Platforms

The front end is implemented and tested on a specific platform, and may not work as intended on others. We develop the front end for a system with the following properties:

Input A mouse and keyboard as input

Screen Resolutions A screen resolution between 1920 × 1080 and 1366 × 768 with the 16:9 aspect ratio

Browser Firefox 42.0 or Chrome Version 47.0.2526.106 m as browser

6.3 Final User Interface Design

The final design, shown in Figure 6.3, generally looks similar to the design presented in Section 2.3. This section presents the final design and describes the differences between the original design and the implemented version.

One of the main differences between the sketched design and the final design is that events located near each other on the map are combined into a group of events when the map is zoomed out. The top red circle on the map over Silkeborg is an example of a group which contains 17 events. Zooming closer the cluster splits the group into smaller groups and eventually into single events as those shown in Figure 6.5.

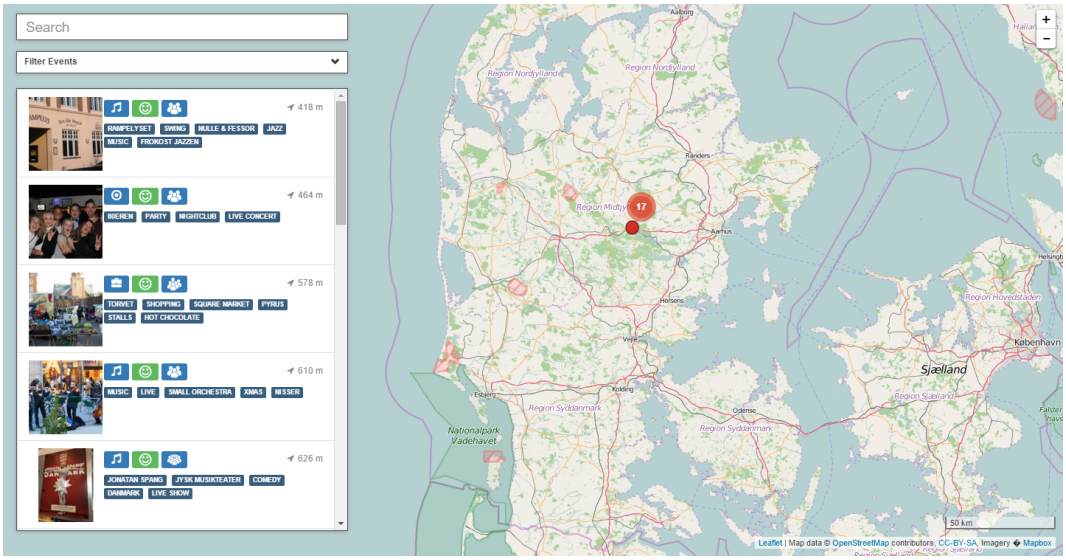


Figure 6.3: Screenshot of the user interface

The search bar, which can be seen in Figure 6.4, generally looks as designed. Clicking on an item in the list moves the map to the location of that item.

In contrast to the original design, the final design does not provide the ability to filter events based on sentiment and size. The filter box, shown in Figure 6.5, only

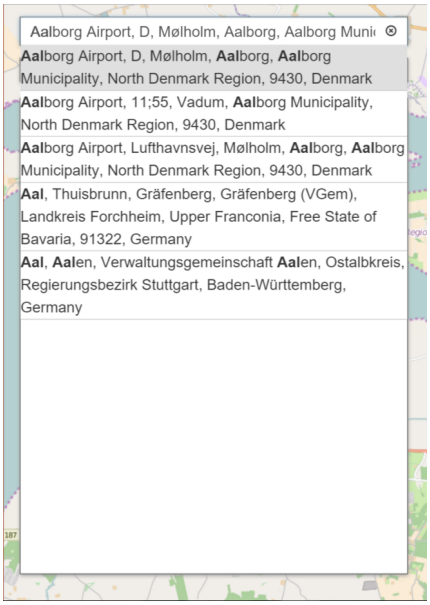


Figure 6.4: The location search bar with a list of matching places

allows for filtering on categories. This deviation is due to limited time available for the implementation. Each category is represented by an icon and a name.

The event list shows the events visible on map. Hovering an event will change the background of the list item and the corresponding marker on the map. The sentiment, size, and category of an event are represented by icons. Hovering these icons show a tooltip which describe them in text, e.g with the name of the category. The background color of the sentiment icon is set according to the sentiment, with green representing positive and red representing negative.

Events can be selected by clicking an item in the list or a marker on the map. A selection replaces the event list with an event description as seen in Figure 6.6b. In contrast to the original design, there is no textual indication of the location of an event. This text is omitted due to limited time. The marker of a selected event is indicated by a blue background color, as shown in Figure 6.6a.

6.4 Usability Evaluation

The purpose of the usability evaluation is to get and understanding of how usable the application is in its current stage, and to identify usability problems. In this section we describe the focus, the setup, and the results of the usability evaluation of the web application.

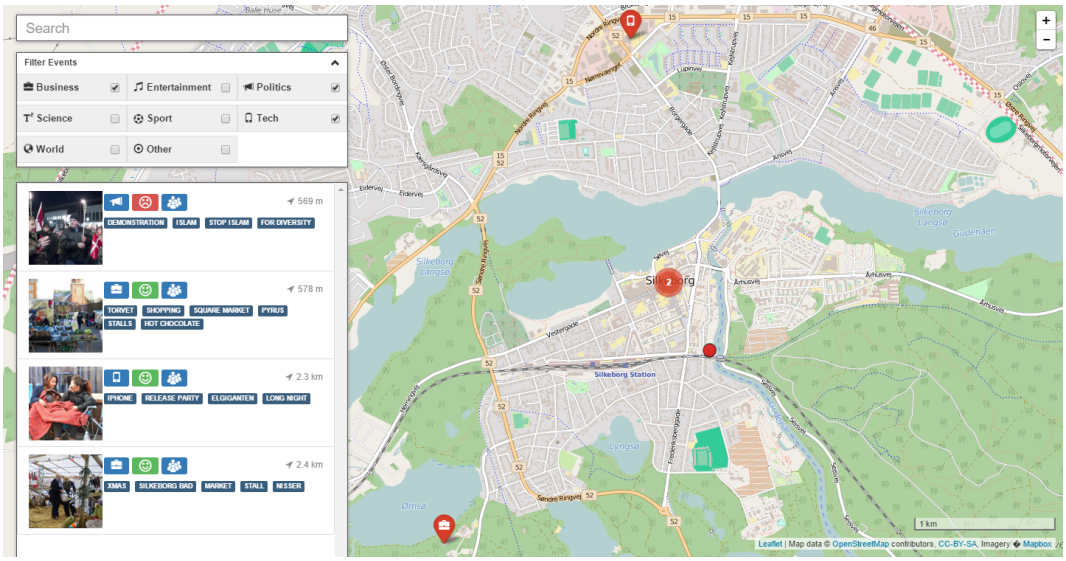
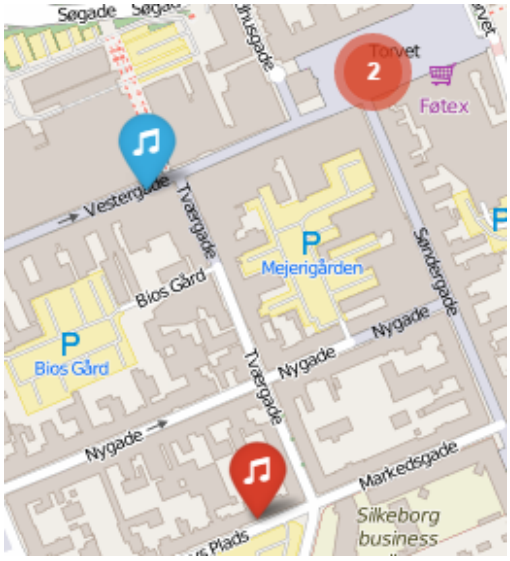
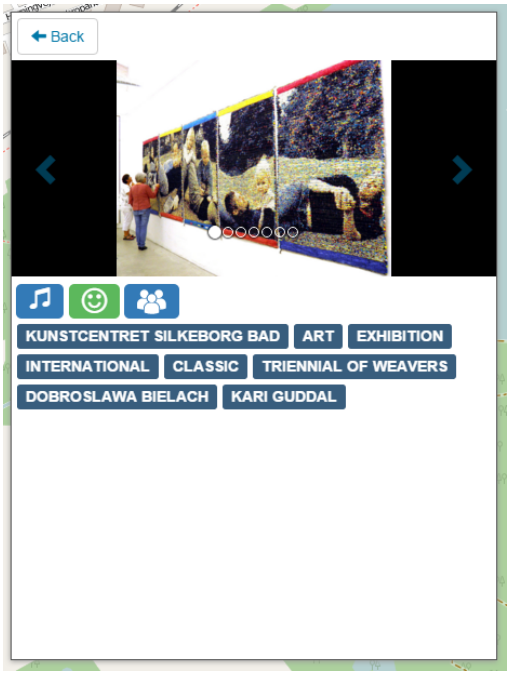


Figure 6.5: Map with some events filtered out



(a) Map with a selected event



(b) Event description of an event

Figure 6.6

6.4.1 Goals and Concerns

We design a usability evaluation which has a high ratio of issues identified compared to the expended effort. As suggested by [12] goals and concerns can be used to design a usability evaluation with a focus on the most relevant aspects of the user interface. The goals specify what we expect the users are able to do in the system, and the concerns are related to elements in the user interface about which we are unsure regarding the usability. It is not possible to have a single evaluation that addresses all the concerns, so only some of them are selected for this evaluation. Based on the chosen goals and concerns we design the evaluation.

Goals

The goals we choose for the usability evaluation do not contain concrete measurements such as time or number of clicks, as we are in an early version of the user interface design. In this stage of the development it has not been presented to users before, and we want to focus more on the users' thoughts and experiences with the system, rather than validate that the use of it is fast. The goals are based on the requirements from Section 2.1, and are what we deem necessary before the system is usable. We state the following goals:

- Users are able to find events close to their own locations with the use of the application
- Users are able to find interesting events with the use of the application
- Users know how to navigate to an event based on the information provided by the application

Concerns

The concerns for the design of the web application are based on our own concerns and thoughts about the usability in the design phase. They represent elements we want to make sure are well designed and if not, how we can make the design better in these specific parts of the system. The user in the following concerns is a person within our target audience, i.e. a young English speaking tourist. The concerns are as follows:

- Will the user be able to find their own location when the application does not provide help?
- Will the user be able to use the search function to find locations?
- Will the user understand what happens when no events are shown, and how to recover from it?
- Will the user be able to filter events by category?
- Will the user understand what the categories cover?

- Will the information provided about events be sufficient for the user to decide if an event is interesting?
- Will some of the presented information be superfluous?

6.4.2 Usability Evaluation Method

We are not usability experts and therefore we cannot do an expert evaluation. Besides, an expert evaluation by us would probably not uncover many new problems, as we are also the developers of the interface. Therefore we choose to perform a usability evaluation on potential users, despite the fact that it takes more time. The evaluation method we choose is an assessment test with some explorative elements. Explorative evaluation is beneficial to use in the early stages of interface design[45].

We consider two different approaches for evaluating the usability based on the assessment test: Video Data Analysis (VDA) and Instant Data Analysis (IDA). VDA is the traditional way to perform usability assessments, and it generates large amounts of data. However, the data analysis is very time consuming and can take up to 40 man-hours for a test with five subjects[32]. The main advantage of IDA is that it demands approximately one tenth of the time than VDA for data analysis, while still identifying as much as 85 % of the critical usability errors and 68 % of the serious problems[32]. Therefore we use IDA for our evaluation.

6.4.3 Setup

In the following sections, we describe the test setup for our usability evaluation.

Test Subjects

When using IDA the appropriate number of test persons is between four and six persons [32]. Our target group is as mentioned young tourist. In addition we have limited the system to only be in English, and therefore it is necessary that the test subjects understand English.

The web application requires the users to have two skills: the ability to use a web browser and the ability to navigate by a map. Ideally, we would like to have test subjects with different levels of experience within both areas. However our target group is typically experienced with the use of computers and web browsing, so it is sufficient that the test subject have different levels of experience with navigating by map. The demographics of the test subjects can be seen in appendix A.1.

Environment

We perform the tests in a usability lab. We have a moderator in the test room with the test subjects, and observers in the observation room. This allows the observers to communicate freely.

Events in Silkeborg
You are on a holiday in Silkeborg, and are staying at Silkeborg Vandrehjem. It is a Saturday afternoon, and you have no plans for the rest of the day. You would like to participate in some activities during the rest of the day.

- Find your own location on the map, i.e. Silkeborg Vandrehjem
- Check if there are interesting events nearby
- Choose an event you would like to participate in
- Get the necessary information, so you can navigate to the event

Figure 6.7: An example task

A computer will run a local version of our application with static evaluation data, in order to get consistency between the evaluation sessions. The data set is artificial and created for the evaluation, to ensure the quality of the presented events, because our focus is on the user interface alone and not the performance of the event detection system.

Evaluation Session

Each evaluation session consists of tree main parts: an introduction, a task execution, and a follow up questionnaire. The format of the session is explained during the introduction, and the test subject can ask questions about the session. In the task execution part the test subject uses the web application to solve predefined tasks. While solving the tasks the test subject thinks aloud as required by the instant data analysis method[32]. Lastly, the session is wrapped up with a questionnaire concerning the test subject's demographic and opinions about the system.

User Tasks

For the usability evaluation we create several main tasks based on the goals and concerns. Each main task includes a description of a realistic use case and concrete tasks for the test subject, as suggested by [45]. An example task can be seen in Figure 6.7.

6.4.4 Results

The analysis of the assessment test results in a list of usability problem ranked by severity. The most severe usability problems, i.e. critical problems, are necessary to resolve before the system is considered usable. We define the severities as follows:

Cosmetic The test subject is delayed for a few seconds

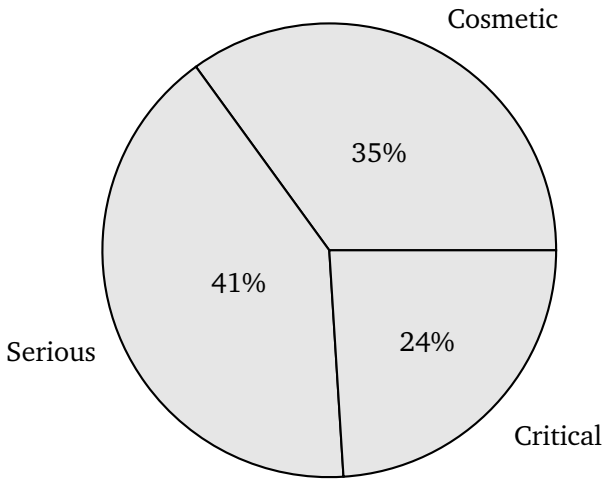


Figure 6.8: Severity of usability problems

Serious The test subject is delayed for more than a few seconds, but eventually accomplishes the task

Critical The test subject is unable to proceed without help

The analysis reveals 17 usability problems, most of them being cosmetic or serious, as can be seen in Figure 6.8. However, there are some critical usability problems. The full list and description of problems can be seen in Appendix A.2. In Table 6.1, we evaluate the fulfillment of the goals mentioned in Section 6.4.1.

Goal	Evaluation
Users are able to find events close to their own locations with the use of the application	All the test subjects are, after some minor difficulties, able to find some events near to their locations.
Users are able to find interesting events with the use of the application	Either the test subjects do not discover all the information about the events or they find it insufficient in order for them to determine if an event is interesting to them. Therefore, we do not consider this goal to be fulfilled.
Users know how to navigate to an event based on the information provided by the application	All the test subjects believe that they are be able to find their way to the event. That being said, they all expect some form of built-in navigational help e.g. in the form of an address or directions.

Table 6.1: Evaluation of usability goals

6.4.5 Discussion

While the usability evaluation uncovers numerous problems with many aspects of the user interface, the test subjects are generally able to accomplish their tasks without too much trouble. An area with severe problems is the way the events are presented. Most of these problems are not related to how the information is presented, but to that the amount of information is insufficient for the test subjects to make a decision. They all request more information, specifically a short summary or description of the event, which consists of complete sentences and not just keywords. Another element all test subjects have issues with is the search box. The most severe problem is caused by the way a search is initiated. When a user begins to type in the box, results appear in the drop down menu. These results are interpreted as suggestions, rather than results, which leads to confusion. The search box also does not include fuzzy matches for a query, which results in no matches if a part of the query is misspelled. All the discovered usability problems can be solved by minor changes to the interface. The terminology and structure of the information appears to be sound, judging by the test subjects' experiences with the user interface.

7 Conclusion

We sought to solve the problem of detecting geospatial events from social media services early enough to find them while they are ongoing. Based on the large amounts of unstructured information on social media, we wanted to separate the posts describing events from the rest. Finally, we wanted to present the detected events to the users.

We have designed and implemented a system which through a multilayered architecture detects events using posts from Twitter and Instagram. The system extracts useful information from the posts and detects ongoing events mentioned in the posts through several stages of clustering and classification.

In the problem statement, we devised the following questions:

- How can events be detected early, so that they can be discovered while they are ongoing?
- How can events be distinguished from non-event data such as breaking news?
- How can events be presented such that young tourists can find those they find interesting?

To detect events early, we only look at social media statuses posted during the latest hour with the cost of having less data available for analysis. In addition, we have designed and implemented a fast event detection pipeline, which runs sufficiently fast to analyze posts while they are still relevant. In our evaluation, all detected events were ongoing at the time they were detected.

In order to distinguish events from non-events such as breaking news, we take an approach which favors sets of posts concentrated in a specific geographical area. Our evaluation shows that the system is able to detect events from the social medium posts. Out of twelve events containing more than five posts, the system detects six. However, the system also detects many false positive events, primarily caused by an inefficient event classifier. A part of this likely stems from the small amount of representative training data for the event classifier. The clustering algorithm also does not do a very good job of clustering the post, only four of the clusters are sufficiently pure to be considered an event. Some of the less pure cluster does still contain enough information for a human to be able to recognize the event. Therefore we might still identify more events than there are pure clusters, as the impure clusters may still be identified as events.

The events found by the event detection system are presented to users through a user interface on a map in a web application. By evaluating the user interface, we found a number of issues that damages the user experience. The most significant problem was insufficient descriptive properties. However, we found that the overall layout and navigation was easy to use.

7.1 Future Work

There are many aspects of this project where we would like to spend more time on improving the system. However, there is a limit to how much time we can to spend during a semester. In this section we describe our ideas for improvement and extension of the system.

7.1.1 Event Detection

The event detection is a large part of the system and while it has received the most attention, there are still many things that could be done to improve and extend it.

Parameter Tuning

The clustering could most likely be improved, simply by spending more time on fine tuning the parameters to both the hashtag clustering and the DBSCAN algorithm. Lowering the minimum size of a cluster to a single post will make the clustering able to accurately cluster all events as some events only contain a single post.

The location clustering which uses DBSCAN could potentially be improved by adjusting the max distance between core points after the population density of different regions. This idea stems from the intuition that events will occur closer and be more concentrated in areas with a high population density.

The filtering of common hashtags can also be improved. The hashtag frequencies we use for calculating idf values are not collected specifically from London. This leads to some almost meaningless but common hashtags being used for the clustering of posts. To improve upon the filtering, there should be regional idf values, such that a term like "London" is not given any significance when the post originates from London.

More Input Features for Event Classifier

The event classifier can be improved in several ways. One way is to provide more input features. The location of an event, time of day, and day of the week, are examples of features that may improve the classifier. These features will allow the classifier to learn specific locations and times as more likely to be the time and place of an event, e.g. music venues, theaters, and stadiums. These extra features will also require more information for training. To properly train the classifier on these features we will need data to be collected for several weeks, and possibly millions of posts to be labeled. It took us about 18 man-hours to label around 9000 posts, gathered during one hour a Tuesday evening in London. During an entire day we collected about 105 000 posts and during an entire week 737 000 posts. At a labeling rate of 500 posts an hour, this equates to roughly 61 man-days of labor. Even when using a service such as Amazon's Mechanical Turk and paying between $0.01 and $0.10 for each label, the cost of labeling that quantity of data will amount to between approximately $7000 and $73 000. Additional work is required to ensure quality of the labeling.

Given a better classifier, we can afford the clustering to let more noise through as well, as long as the purity of the event clusters remains high.

Other Data Sources

There are other data sources which we do not use, that we would like to expand to. Swarm is used by people to check in to locations and could be used as an additional source of posts. Facebook is another option. We could develop a Facebook application which would allow users to check in through Facebook, and make us capable of using the status updates of our users as another source of posts.

We do not use the user profile information in our event detection. If we include the user-id as an input feature, we might be able to assign users a trustworthiness score of sorts. User accounts used for spam could be removed entirely, and users who often go out and post about it would get a higher trustworthiness score.

We can also extract more information by following links embedded in posts and try to reason about what kind of content the link points to. This could also help determine whether the post is from an event. Many posts also contain pictures which could be analyzed to gather more information about the posts.

7.1.2 Front End

During the usability evaluation we discovered a number of problems which we would like to resolve. There are also several other areas in which the presentation of detected events can be improved, which makes the overall system more useful for users.

Event Description

One issue with the current approach is the lack of information about an event. The keywords are generally imprecise. It would be interesting to try applying a more complex way of determining the topics of an event, such as a topic model like *Latent Dirichlet Allocation* [6]. Instead of relying on informative hashtags in the post, such a model can be used to assign several topics to an event. Another and more simple option is to show a sample of the posts used for detecting the event, though the quality of this clearly depends on the quality of sampled posts, and not on the posts as a whole.

Responsive User Interface

While the current user interface works well on a desktop computer, it does not work well on devices with smaller screens such as smartphones. Because it is likely that users look for events while they are on the move, the user interface should be made responsive such that it works on most types of devices. The front end could potentially also be developed as a smartphone application.

Personal Event Recommendations

To discover events that a user finds interesting, they will have to filter on the predefined categories. This provides little customizability and may make it difficult for a user to find events. An interesting improvement would be to learn the interests of a user and automatically recommend events based in these interests. In combination with a smartphone application, one could imagine that the system notifies a user about interesting events happening near their current location.

7.1.3 Scalability

While we have attempted to build an efficient analysis pipeline, the currently implemented architecture does not directly scale to work with significantly larger data streams such as the Twitter Firehose. In order to handle more data, the system should distribute posts across multiple servers. We suggest that the data is distributed based on geographic location of posts. One such approach would be to divide the Earth into a number of overlapping rectangles, such that only the posts within one rectangle are considered when doing event detection. By using sufficiently large overlaps and rectangles, merging potential events that overlaps several rectangles is easy by assuming that no event is larger than one rectangle. In this case, all events will be entirely represented within one of the rectangles. One rectangle need not correspond to one server; depending on the activity within the rectangle, one server may be able to handle several rectangles. Alternatively, rectangles could scale according to the activity level within the geographical regions they represent.

8 Reflection on Development Process

In this chapter, we present and reflect on the development method used throughout the creation of this project.

8.1 Process Description

This project utilizes many new areas of theory and practice for us, mainly related to information retrieval and web intelligence. This makes it difficult to plan the development, and large parts of the project will likely change over time as we gain more knowledge in the area. It is not possible for us to learn the necessary theory and gain significant experience in the beginning of the project, and yet have time to design the event detection system. Because of these uncertainties, we choose an agile development process to accommodate that we refine our knowledge over time and learn new techniques. In addition, our focus in this project is fairly research oriented, and we focus on experimentation with and exploration of different approaches to find a suitable way of solving the problem. It is a relatively short project period, and we assess that we do not need a lot artifacts to help ourselves during the development. We therefore choose a relatively lightweight development framework. As we have prior experience with the Scrum framework, we use Scrum as the foundation for our process and tailor it for our project. Throughout the project, we continually refine our development method to our needs.

8.1.1 Roles

We choose to have no named Product Owner and no named Scrum Master. The tasks and responsibilities normally assigned to these roles are shared in this project. Since we have no external customer, we can prioritize the features together and decide on what is most important for our project. In addition, it is shared responsibility to ensure everyone is fully functional and productive in team. Our team consist of five people, and we predominantly work in the same room to ensure easy communication and knowledge sharing.

8.1.2 Ceremonies

We start every sprint with a sprint planning where we discuss the focus or goals for the sprint. We decide on which tasks are necessary to reach the goals. We play planning poker to estimate the time needed for each task and if necessary, we adjust the goals of

the sprint so the amount of work fit the available time in the sprint. Tasks are estimated in half workdays, which corresponds to 3.5 hours. We choose this as our time unit as we expect that we cannot estimate tasks more accurately and because it fits with the university schedule.

At the end of a sprint, we hove a sprint review and a sprint retrospective. In the sprint review, we give a short summary of the work finished in the sprint. Because we only have the meeting for ourselves with no external people, we can keep the meeting very short and informal as everyone in the team should already know most of the particulars. After the sprint review, we evaluate on the progress made in the sprint and revise the method as needed.

We have a stand-up meeting daily on workdays, where we give status on work and commit to tasks. If a new task which is necessary to achieve the goal of the sprint occurs, it is added on a task board on these meetings.

8.1.3 Artifacts

In the development process, we use a product backlog, a kanban board, and a burn down chart. The product backlog contains user stories which describe the features we need to implement. As we have no customers, the user stories are written based on what we think are necessary elements in a solution to the problem. For each sprint we select a subset of the user stories, and we add more to the product backlog as we learn more about the problem area.

During each sprint, we use the kanban board to keep track of our tasks and the burn down chart to measure our progress. The kanban board contains *todo tasks*, *work in progress tasks*, *done tasks*, and *done done tasks*, where *done* means that a task is completed but needs a review by another person and *done done* means that a task has been approved.

8.1.4 Development Environment

We establish a test environment in the beginning of the development process to make it possible to test and debug the system on reproducible data. This makes it possibly to use the system without subscribing to live data from social media but instead use posts from a database.

In addition, we use a virtual environment for Python to keep consistency in our local Python environments and to make sure everyone is using the same versions of the libraries we use. This also ensures that our local programs can be used on a server.

8.2 Process Reflection

Generally, we found the use of a light, agile method a good fit for this project. However, we had some difficulties with time pressure in the end of the project, due to different problems with the process and some unexpected challenges. In this section, we will reflect upon the different choices in the process and describe what we should do differently in a future project.

8.2.1 Roles

We have had some difficulties with handling member absence in the project. People often arrived later than agreed, and it has become more the norm than the exception. This of course resulted in less time for work and thus caused difficulties when estimating. This could potentially have been be remedied by having a person assigned as Scrum Master, who had the formal responsibility that everybody could work productively.

There was additional absence leading up to a (former) member leaving the team, which we should have addressed earlier in the project. Because this happened a few weeks before handing in the project, we decided to increase the number of working hours instead of limiting the scope of the project.

8.2.2 Ceremonies

Despite agreeing otherwise, the daily stand-up meetings turned out not to occur at a fixed time every day. The main reason for this was that people often arrived late, which caused us to postpone the meeting. At the time before the meeting, people sometimes did not know what to work on, and it was difficult to estimate whether there was time to start on a new task. The meeting could possibly be placed later on the day, or we could have handled the problem of late arrivals internally.

During sprint plannings, we found it difficult to estimate tasks as a lot of them included unknown factors. This caused many tasks to be defined without an expected time, which was unfortunate as it made it difficult to estimate whether the project was on track and people who committed to a task did not feel a responsibility to finish on time. Generally, we were not sufficiently attentive on whether tasks took longer time than estimated and what the reason was. To avoid this problem, we could have spent more time during the planning meetings for analyzing the extend of different tasks.

During the sprint reviews, we where not sufficiently attentive to indications that we were behind schedule. Even though the burn down chart showed that we where behind schedule, the signals were ignored and incomplete tasks were transferred to the next sprint. By going through a small checklist of necessary attention points under the meetings, some of the problems may have been addressed earlier.

We found that the time unit of half a man-day worked well, though we sometimes had tasks significantly smaller than half a day. This caused us to over-estimate small tasks, and we expect that allowing a “one hour” time could have solved the problem.

8.2.3 Artifacts

In the beginning of the project, we used the product backlog to keep track of user stories. However, we stopped using it shortly into the project as it seemed more of a burden than a helpful tool. We did not have backlog items for other sprints than the current one, as we did not know what to work further on (due to the experimental approach to the project). The kanban board showed to be sufficient to keep track of the necessary work. However, the backlog could have been used to keep track of the more general goals of

the sprint, and if it was in physical form instead of digital, we may have benefited more from it.

We found the kanban board to be very useful and it provided a good overview of the concrete work we had to do. In addition, the kanban board made it easy to keep track of who where working on which tasks, and it was easy to find new tasks to work on.

While we regularly updated the burn down chart, it was not used as much as desired as it turned out that we did not pay sufficient attention to whether our project was on track. In the future, we should make it a clear routine to check this chart during the daily meeting.

8.2.4 Development Environment

While we had a setup for testing the system, it was not sufficient for being able to evaluate the overall system regularly. The primary cause was a missing labeled dataset, which did not become part of the evaluation setup before the last month of the project. By evaluating on this data earlier, we could have verified approaches during the development, and not first when it was too late to fix severe problems. In the future, we should use more time on making a good evaluation environment early in the project to improve the overall development process.

Bibliography

[1] Ahmed Abbasi, Ammar Hassan, and Milan Dhar. “Benchmarking Twitter Sentiment Analysis Tools”. In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC’14)*. 2014, pp. 823–829.

[2] *Apache Lucene*. Apache Software Foundation, 2015. URL: https://lucene.apache.org/.

[3] *Apache Solr*. Apache Software Foundation, 2015. URL: http://lucene.apache.org/solr/.

[4] Austin Appleby. *MurmurHash3*. 2011. URL: https://code.google.com/p/smhasher/wiki/MurmurHash3.

[5] H. Becker, M. Naaman, and L. Gravano. “Beyond trending topics: Real-world event identification on Twitter”. In: *Fifth International AAAI Conference on Weblogs and Social Media*. 2011.

[6] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. “Latent Dirichlet Allocation”. In: *J. Mach. Learn. Res.* 3 (Mar. 2003), pp. 993–1022. ISSN: 1532-4435.

[7] Andrei Z. Broder et al. “Min-wise Independent Permutations”. In: *Journal of Computer and System Sciences* 60 (1998), pp. 327–336.

[8] Sven Casteleyn et al. *Engineering Web Application*. Springer, 2009. ISBN: 978-3-540-92200-1.

[9] Yann Collet. *xxHash*. 2015. URL: http://cyan4973.github.io/xxHash/.

[10] Sanjiv R. Das and Mike Y. Chen. “Yahoo! For Amazon: Sentiment Extraction from Small Talk on the Web”. In: *Manage. Sci.* 53.9 (Sept. 2007), pp. 1375–1388. ISSN: 0025-1909.

[11] Maeve Duggan et al. *Social Media Update 2014*. Accessed December 19 2015. Pew Research Center, Jan. 2015. URL: http://www.pewinternet.org/files/2015/01/PI_SocialMediaUpdate20144.pdf.

[12] Joseph S. Dumas and Janice C. Redish. *A Practical Guide to Usability Testing*. Revised Edition. Intellect Books, 1999. ISBN: 1-84150-020-8.

[13] Martin Ester et al. “A density-based algorithm for discovering clusters in large spatial databases with noise”. In: AAAI Press, 1996, pp. 226–231.

[14] Facebook. *Facebook groups*. English. Accessed September 29 2015. Facebook. Sept. 2015. URL: https://www.facebook.com/groups/4328544524/.

[15] Facebook. *facebook.com/robots.txt*. English. Accessed September 22 2015. Facebook. Sept. 2015. URL: https://www.facebook.com/robots.txt.

[16] *Facebook Upcoming Events*. Accessed September 22 2015. Facebook, 2015. URL: www.facebook.com/events/upcoming.

[17] *ForgetIT Project*. ForgetIT Project, 2015. URL: http://www.forgetit-project.eu/.

[18] Alec Go, Richa Bhayani, and Lei Huang. *Sentiment140*. English. Accessed December 7 2015. Dec. 2015. URL: http://help.sentiment140.com/for-students.

[19] Alec Go, Richa Bhayani, and Lei Huang. "Twitter sentiment classification using distant supervision". In: *CS224N Project Report, Stanford* 1 (2009), p. 12.

[20] Ralph F. Grove and Eray Ozkan. "The MVC-web Design Pattern". In: *WEBIST 2011, Proceedings of the 7th International Conference on Web Information Systems and Technologies, Noordwijkerhout, The Netherlands, 6-9 May, 2011*. 2011, pp. 127–130.

[21] Jeff Huang, Katherine M. Thornton, and Efthimis N. Efthimiadis. "Conversational Tagging in Twitter". In: *Proceedings of the 21st ACM Conference on Hypertext and Hypermedia*. HT '10. Toronto, Ontario, Canada: ACM, 2010, pp. 173–178. ISBN: 978-1-4503-0041-4.

[22] Kenneth F. Hyde. "Information processing and touring planning theory". In: *Annals of Tourism Research* 35.3 (2008), pp. 712–731. ISSN: 0160-7383.

[23] Instagram. *Instagram API Limits*. English. Accessed September 23 2015. Instagram. Sept. 2015. URL: https://instagram.com/developer/limits/.

[24] Instagram. *Instagram Help*. English. Accessed September 22 2015. Instagram. Sept. 2015. URL: https://help.instagram.com/182492381886913/.

[25] Instagram. *instagram.com/robots*. English. Accessed September 22 2015. Instagram. Sept. 2015. URL: https://www.instagram.com/robots.txt.

[26] Instagram. *Media Endpoints*. Accessed October 13 2015. Oct. 2015. URL: https://instagram.com/developer/endpoints/media/.

[27] Instagram. *Real-time Photo Updates*. Accessed October 13 2015. Oct. 2015. URL: https://www.instagram.com/developer/deprecated/realtime/.

[28] Akshay Java et al. "Why We Twitter: Understanding Microblogging Usage and Communities". In: *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis*. WebKDD/SNA-KDD '07. San Jose, California: ACM, 2007, pp. 56–65. ISBN: 978-1-59593-848-0.

[29] Mads Ellersgaard Kalør et al. *Et nyhedsfilter*. 2013.

[30] Andreas M. Kaplan and Michael Haenlein. "Users of the world, unite! The challenges and opportunities of Social Media". In: *Business Horizons* 53.1 (2010), pp. 59–68. ISSN: 0007-6813.

[31] Adam Kirsch and Michael Mitzenmacher. "Less Hashing, Same Performance: Building a Better Bloom Filter". In: *Random Struct. Algorithms* 33.2 (Sept. 2008), pp. 187–218. ISSN: 1042-9832.

[32] Jesper Kjeldskov, Mikael B. Skov, and Jan Stage. *Instant Data Analysis: Conducting Usability Evaluations in a Day*. Oct. 2004.

[33] Raffi Krikorian. Accessed December 19 2015. 2013. URL: https://blog.twitter.com/2013/new-tweets-per-second-record-and-how.

[34] Ryong Lee and Kazutoshi Sumiya. "Measuring Geographical Regularities of Crowd Behaviors for Twitter-based Geo-social Event Detection". In: *Proceedings of the 2Nd ACM SIGSPATIAL International Workshop on Location Based Social Networks*. LBSN '10. San Jose, California: ACM, 2010, pp. 1–10. ISBN: 978-1-4503-0434-4.

[35] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. 2nd ed. Cambridge University Press, 2014. ISBN: 9781107077232.

[36] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008. ISBN: 0521865719, 9780521865715.

[37] Philip J. McParlane, Andrew J. McMinn, and Joemon M. Jose. "*Picture the scene...*"*; Visually Summarising Social Media Events*. 2014. URL: http://www.dcs.gla.ac.uk/~philip/papers/8.pdf.

[38] Geo MidPoint. *Geo Midpoint Calculation Methods*. Accessed December 19 2015.

[39] Sarah Mitroff. *Swarm review*. English. Accessed September 24 2015. CNET. May 2014. URL: http://www.cnet.com/products/swarm/.

[40] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. "Thumbs Up?: Sentiment Classification Using Machine Learning Techniques". In: *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*. EMNLP '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 79–86.

[41] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[42] Christopher Potts. *Sentiment Symposium Tutorial: Stemming*. Accessed November 18 2015. 2011. URL: http://sentiment.christopherpotts.net/stemming.html.

[43] PRNewswire. *Facebook Reports Third Quarter 2015 Results*. Menlo Park, CA, USA: Facebook, Nov. 2015. URL: http://files.shareholder.com/downloads/AMDA-NJ5DZ/1197106350x0x859021/F783FA3F-CB65-42C2-AB30-5F4C56567A31/FB_News_2015_11_4_Financial_Releases.pdf.

[44] Hope Restle. *Here's who is using Twitter around the world*. English. Accessed September 22 2015. Business Insider UK. June 2015. URL: http://uk.businessinsider.com/who-uses-twitter-2015-6?r=US&IR=T.

[45] Jeff Rubin and Dana Chisnell. *Handbook of Usability Testing, Second Edition: How to Plan, Design, and Conduct Effective Tests*. 2008. ISBN: 978-0-470-18548-3.

[46] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2nd ed. Pearson Education, 2003. ISBN: 0137903952.

[47] 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*. WWW ’10. Raleigh, North Carolina, USA: ACM, 2010, pp. 851–860. ISBN: 978-1-60558-799-8.

[48] Nakatani Shuyo. *Language Detection Library for Java*. Accessed November 18 2015. 2010. URL: https://github.com/shuyo/language-detection.

[49] Nakatani Shuyo. *Language Detection Library for Java*. Accessed November 18 2015. 2010. URL: http://www.slideshare.net/shuyo/language-detection-library-for-java.

[50] Taylor Singletary. *Is there a limit to the amount of data the streaming API will send out?* Accessed September 24 2015. Mar. 2012. URL: https://twittercommunity.com/t/is-there-a-limit-to-the-amount-of-data-the-streaming-api-will-send-out/8482.

[51] Craig Smith. *By the Numbers: 50+ Amazing Google+ Statistics*. English. Accessed September 22 2015. DMR. July 2015. URL: http://expandedramblings.com/index.php/google-plus-statistics/.

[52] statista. *Leading social networks worldwide as of August 2015, ranked by number of active users (in millions)*. English. Accessed September 22 2015. statista. Sept. 2015. URL: http://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/.

[53] Mitja Trampus. *Evaluating language identification performance*. Accessed November 25 2015. 2015. URL: https://blog.twitter.com/2015/evaluating-language-identification-performance.

[54] A. Tumasjan et al. “Predicting elections with twitter: What 140 characters reveal about political sentiment”. In: *Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media*. 2010, pp. 178–185.

[55] Twitter. *Twitter Documentation*. English. Accessed September 18 2015. Twitter. Sept. 2015. URL: https://dev.twitter.com/overview/documentation.

[56] Maximilian Walther and Michael Kaisser. “Geo-spatial Event Detection in the Twitter Stream”. In: *Proceedings of the 35th European Conference on Advances in Information Retrieval*. ECIR’13. Moscow, Russia: Springer-Verlag, 2013, pp. 356–367. ISBN: 978-3-642-36972-8.

[57] Yiming Yang et al. “Learning Approaches for Detecting and Tracking News Events”. In: *IEEE Intelligent Systems* 14.4 (July 1999), pp. 32–43. ISSN: 1541-1672.

[58] Kathryn Zickuhr. *Location-Based Services*. Accessed September 18 2015. Pew Research Center, Sept. 2013. URL: http://www.pewinternet.org/2013/09/12/location-based-services.

A Usability Evaluation Results

A.1 Demographic of Test Subject

The demographic of the test subjects for the usability evaluation can be seen in table A.1.

A.2 Usability Problems

A.2.1 Cosmetic Problems

Confusing map The map contains too much information/details, e.g. direction on road momentarily mistaken for navigation clues.

Event list unclear When the event list is empty at first, the purpose of it is unclear.

Missing feedback from search input It is not clear for the user what happens, when a search query return no results.

Closing events description The user tries the close the event by clicking on the event again or elsewhere on the map

Event outside list bug At some point the user discovered a bug which made an event appear below the event list.

Event clusters can be misleading Some users think that the clustered events are enumerated events instead of clusters of events.

A.2.2 Serious Problems

Search result not clear The result of a search is not clear enough. The user cannot see the place they have search for. For example a search for "Tange" moves map, but if "Tange" is not written on the map, the user thinks the position of the map is wrong.

	Age	Use computer to browsing	Experience with map to navigation	English level
Test subject 1	23	Every day	Have tried it	Practical proficiency
Test subject 2	22	Every day	Have tried it	Limited proficiency
Test subject 3	23	Every day	Good at it	Practical proficiency
Test subject 4	25	Every day	Good at it	Fluent

Table A.1: Demographic of test subjects

Missing Route planning The possibility of step-by-step route information was expected, and the user spends a lot time on finding information enough to navigate without.

Bad front page The user got confused because no information were given about the site or how it worked. The map as first page was confusing.

Missing marked event When a user enters an event description and moves the map away from it they have problems locating it again.

Misleading icons Some icons does not match the event description. It is confusing when a 'Note' i.e music also means other types of entertainments.

Unclear event list error message It is unclear to the user what is wrong, when there are no events found in a location. The error message is read, but first after a while, and the users does still not know why there is no events.

Move "You are here" marker The user tried to move the "You are here" marker to a different location if it does not match their location.

A.2.3 Critical Problems

Bad search function The user have problems using the search bar. E.g. it has limited search options and can't handle spelling errors, therefore the user cannot find the wanted locations.

To little event information Too little information of the event was available and the user could not decide if and event was interesting.

Bug when filtering Several users discovered a technical problem when unchecking events in the event filter function. The event does not disappear but when rechecking it, it does.

Missing system info More information about the different parts of the system like e.g. what a category covers and how sentiment and size is calculated. This would make the site more trustworthy according to a test subject.