Wageningen University
Laboratory of Geo-Information Science and Remote Sensing

Multi-Dimensional Event Detection on Twitter

CAS knowledge discovery using Big Data Analysis

CODE NR: GRS-80436 , REPORT NR: GIRS -2012 -20

By

Boyen S. van Gorp

Multi-Dimensional Event Detection on Twitter

CAS knowledge discovery using Big Data Analysis

Boyen S. van Gorp

861015271090

Supervisor:

Arend Ligtenberg

A thesis submitted in partial fulfilment of the degree of Master of Science

at Wageningen University and Research Centre,

The Netherlands.

14th of April 2014

Wageningen, The Netherlands

Thesis code number: GRS-80436
Thesis Report: GIRS-2012 -20
Wageningen University and Research Centre
Laboratory of Geo-Information Science and Remote Sensing

Contents

List of Figures

List of Tables

List of codes

Acknowledgements

I wish to thank first and foremost my supervisor, Arend Ligtenberg, who not only showed the ability to understand my often chaotic stories but also has the ability exactly pinpoint the weaknesses in my theory. The many hours of critical discussion improved the quality tremendously.

I would also like to thank Mark Kramer, who helped a great deal by providing theoretical background on how to implement a Big Data project.

In addition, my gratitude to Lieke Melsen. Not only for the many brainstorm sessions, but also for introducing and guiding me in the world of LaTeX.

A special thanks to Ron Zacharski who introduced me to the wonderful world of data mining. Without his help I could not have constructed the methodology as it is now.

Last but not least I would like to express my gratitude to Elyn den Hollander, for being a continuous support and always asking the right questions.

Abstract

Twitter is an easily accessible geo referenced data platform which has a large potential for geo-information science. Many researchers have made attempts to extract events from Twitter with mixed results. Ad hoc definitions are used to solve specific problems without attention for the overall problem. A paradigm which serves as theoretic background for event detection from social media is missing. This research attempts to provide a definitions for "events" and "Twitter" leading to an objective polymorph methodology that can be applied regardless of context.

A combination of cross-disciplinary definitions is used to form an overarching event definition. Events are identified as multi-dimensional collection of happenings that have four dimensions: *context*, *spatial* ,*temporal* and *actor*. As such, event detection needs to consider the multi-dimensionality of events within its approach. Each dimension has properties which are identified as a: *centroid*, *volume* and *clustering*. This research lays a focus on events in the digital reality and attempts to create a methodology that is analogous to event detection in the physical reality in the human brain. In which happenings of similar dimensionality are clustered together forming events.

Twitter is classified as a Complex Adaptive System (CAS), which gives implications on its features and behaviours. Assumptions of homogeneity and linearity are not fitting in the context of a CAS which requires a complex analysis approach. Big Data analysis is an especially suitable method to analyse Twitter as it allows describing complex events in great detail. However, to reach a knowledge level in which the data can be described, several steps need to be taken. A methodology is created for the first three steps following the BSID knowledge pyramid: *procurement* (Big Data), *clustering* (Small Data) and *classification* (Information) of the data.

The implementation of the *procurement* of Big Data (Tweets) is done using the Twitter REST API in a Python interface. Tweets are saved in a MongoDB which is chosen due to its flexibility in queries. A flexible querying structure is essential as it allows efficient selections on the data saving computing resources. The step from Big Data to Small Data is made by *clustering* using the *K-Means* algorithm, with optimized initial centroids, in both the spatial and context dimension. Small Data is turned into Information by *classification* of the created clusters, using the properties of the dimensions in line with the event definition. The *clustering* of a collection is measured using the *k-Nearest Neighbour (kNN)* and *k-Cosine Similarity (kCS)* algorithms for the *spatial* and *context* dimension respectively. The context dimension is normalized by creating a vector space model *term frequency (tf) inverse document frequency (idf)*.

Event collections are often built up using the Twitter structuring mechanisms (hashtags, mentions). Even if alternative methods are used the data is not classified, which leads to uncertainty on the contents of the data. The proposed methodology solves these issues by: identification of event clusters in multiple dimensions, event clustering between/within collections improving the value of the data and an objective multi-dimensional classification methodology.

1 Introduction

1.1 Twitter and Big Data

Twitter is a social media micro blogging platform[25]. Compared to normal blogging it has a low time investment and is feasible on mobile devices which leads to more frequent posting compared to normal blogs (hours vs. days). Leetaru et al.[31] found that about 2% of all Tweets contain geographic meta data. This number seems small, but considering a volume of 9100 Tweets each second[49] even the small percentage of Tweets that are geo-tagged has the potential to be an interesting source of geo-information data[53]. Twitter is also a convenient platform to analyse due to several common practices that structure Tweets (see Figure 1) and an API that allows easy access to the data[52].

Due to its velocity and large volumes of data, Twitter can be considered a Big Data source. The term "Big Data" indicates large volumes of low-value data which can be used to obtain high-value data. The high-value data has applications in science, technology and business purposes[19]. Though often an ad hoc definition of Big Data is used, usually a combination of the three V's (Volume, Velocity and Variety) are used to describe Big Data (see Figure 2). Big Data analysis (BDA) has the underlying assumption that the data has the potential for describing complex phenomena which can not be done with conventional methods. For example; game companies analysed behaviour of their customers. This analysis allowed focused adjustments of their products, playing into the needs of their customers and making their games a more enjoyable experience[19]. E-commerce uses Big Data analysis to align advertisements with consumer interest by tracking consumer behaviour. Sport organisations use BDA to optimize scheduling of matches to increase income[28] and BDA is used to predict the stock market fluctuations from Twitter mood[9]. The challenges with Big Data do not lie in acquisition of the data, but rather the storage of large volumes of data, accessibility and analysis. Big Data can be re-used and shared to find other complex relationships within the data, enabling a high value gain out of a single investment of data[33].

Twitter can be used as a model for society, though Twitter only represent part of the population. It is mostly visited and used by people of ages between 18-44 while being under-represented in the 44-65+ category[2]. PearAnalytics categorized 80% of all Tweets as "pointless babbling" or "conversational"[27]. However, these numbers change drastically with different categorization. Often an ad hoc approach is taken when defining categories and events making results difficult to interpret objectively. Kwak et al.[30] categorized the content of trending topics as predominantly news or headline related. As such, Twitter has potential applications as a news resource. In their research Twitter was compared to CNN in which especially events of broadcasting nature were reported faster on Twitter. A downside is that the legitimacy of reported events are hard to verify, though a self correcting mechanism concerning false rumours has been observed by Procter et al.[42]. Sakaki et al.[46] proposed a Twitter data analysis method that can be used to track disasters such as Earthquakes. Their method converts semantic properties (context) into events. These event clusters are used to pinpoint the geo-spatial location of the disaster.

1.2 Problem Statement

Event detection is a common theme in Twitter data analysis[42;46;35;14;61;54;3;53;62] (further explored in subsection 2.2), though there is a lack of uniformity in definition and methodology. An ad hoc definition is used for the term "event" and only the dimensions that are of interest for the research are analysed. Ad hoc definitions make it difficult to compare results between research and hinder knowledge development in the field. An overarching definition

Figure 1: Twitter structuring mechanisms. An "@" with a user name also called a mention - implies that the tweet is addressing another user. A "hashtag" is Twitters methodology for grouping Tweets together based on the topic. It is used by typing the hashtag followed by the topic behind it. Hashtags are used to identify trending topics, of which an overview can be found on the website. A retweet implies that the user is sharing a tweet of another user, it allows messages to be spread beyond the followers of the initial tweet, 75% of the retweets happen within the same day[30]. This retweeting mechanism is one of the main drivers behind making a topic go viral as the reach of the initial message grows exponentially by the use of a retweet.

backed up by theory allows standardized methodologies for event detection, while also giving a theoretical underpinning on how events should be detected. This is especially important for automated approaches using algorithms where a different definition leads to different results, even though the goal might be identical.

The most commonly analysed aspects of Tweets with the goal to detect events are the text and the time. The location from which a Tweet originated or which the Tweet concerns is barely explored as a factor for event detection (further explored in subsection 2.2).

The assumption of homogeneity and completeness of Tweets belonging to the same hashtag is a weakness in related works. Users of Twitter can use such hashtags for different purposes and not every user of Twitter will properly use the hashtag mechanism. As such, there is a chance that the data is either incomplete or irrelevant to the analysis. This is an indication that more advanced methodologies than just the common practices of Twitter (Figure 1) should be used to structure the data.

For Event Detection identification of clusters is the first step. The meaning of such clusters is of importance to interpret the data. An objective methodology for classification of data gives context to the clusters. Without this classification there is a large uncertainty on what is analysed. In addition, classification allows grouping of similar clusters giving insights on the relation between clusters.

1.3 Goal & Research Questions

The goal of this research is to develop a methodology for event detection and classification on Twitter. An important aspect of this methodology is that it should be capable of detecting generic events. Generic events detection in this context is event detection that is not specified for a use case. A requirement of the methodology is that it is polymorphic in nature and as such can be easily expanded upon. The following research questions are defined:

(i) How can Twitter event detection be approached to establish a uniform methodology suitable for generic event detection?

1. Which approach should be used to analyse Twitter?
2. How can the term "event" be defined to cover all facets of events, suitable for event detection?

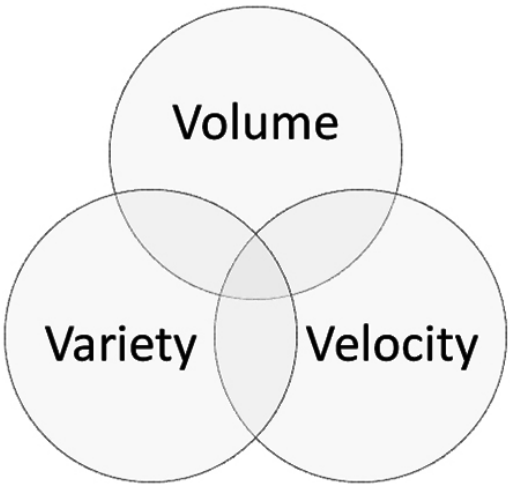

Figure 2: Big Data can be seen as a combination of at least two of three characteristics. *Volume*, which indicates that the amount of data stored is large enough to require a different approach in storage and analysis. *Velocity*, which means that Big Data is produced at high rates. The *Variety* indicates that Big Data often deals with different types of data at the same time, photos, text and videos for instance. It also indicates high dimensional data.[21;15].

3. How can the approach (*RQ1*) and definition (*RQ2*) of an event be translated in a polymorphic methodology for event detection and classification?

In related research Twitter has not been conceptualized by the use of a paradigm. By answering *RQ1* a stronger theoretical underpinning of the methodology can be achieved (explored in subsection 2.1). Event definitions are varied between researchers, a definition for the term event (*RQ2*) is found by analysing definitions in other research in the fields of Event Detection and Psychology. The combination of definitions will be used to form a comprehensive definition (explored in subsection 3.1). The translation from approach and definition to methodology is explored in *RQ3*. The analysis approach as a result of *RQ1* is used to determine the type of analysis. The definition of "event" as a result of *RQ2* is used as a guideline to form a methodology for event detection (explored in subsection 3.2). Based on the strengths and weaknesses found in related works requirements of the methodology will be formulated. The methodology will be tested to establish whether it fulfils these requirements.

2 Background

2.1 Knowledge Discovery

Society can be seen as a Complex Adaptive System (CAS)[12], Twitter, as an exponent of society is likely to be a CAS as well. CAS theory is used in many fields to get a better understanding of the requirements and approach to analyse and understand a system. Grus et al.[22] proposed a framework to identify a CAS using features and behaviours: *components*, *complexity*, *sensitivity to initial conditions*, *scale independence*, *openness*, *unpredictability*, *non-linearity*, *self organizing*, *adaptability* and *feedback loops*. Twitter is build up from simple *components* that are linked together and interact, these components are the users. The *complexity* means that there is a constant exchange of information and needs between components and that the system is more than the sum of its parts. On Twitter there is a constant communication between users. These communications and interactions create a far more complex system compared to a system without these interactions. Twitter is an *open system* that is sensitive to outside influences, users Tweet about Twitter, but also Tweet about events outside the Twitter domain and are affected by these events. Twitter is an *unpredictable* medium, though correlations can be drawn to outside influences, these outside influences are unpredictable as well and the effect of these influences is often *non-linear*. Twitter is a medium used for *self organization*, with no higher authority that manages or plans. This self organizing behaviour was especially apparent in the major role Twitter played in the revolts of the Arabian spring[23]. The components of Twitter are capable of *adapting* influenced by either internal or external factors, the self correcting mechanism as found by Procter et al.[42] is an example of this. The RT and @ mechanisms from Twitter allow *feedback loops*. Twitter is a system that is *sensitive to initial conditions*. Different interactions and phenomena take place on Twitter when compared to other social media. Facebook for instance, had different initial conditions which led to a different velocity of posts, type of posts and interactions between users.

The only mismatch with a CAS is the *scale independence*. Twitter does not seem to be *scale independent*, as no real fractal structure can be observed in Twitter. Overall according to the framework proposed by Grus et al.[22], Twitter is either a CAS or has large similarities to a CAS. This implicates that it should be analysed as a CAS requiring a complex approach (see Table 1).

A wide array of complex analysis methodologies exist. Complex phenomena can be modelled (e.g., using agent based models) but in the case of Twitter the need to model the complex phenomena is less important as data for analysis is generated every second. Twitter produces high velocity streams of large volume, high dimensional unstructured data. It can be classified as a Big Data source (see Figure 2). As such, BDA methodologies could be used to describe the complex phenomena on Twitter. However, it should be noted that no prior reports have been identified that link CAS and BDA.

Knowledge Discovery & Datamining (KDD) is defined as "non trivial extraction of implicit, previously unknown and potentially useful information", the analysis of Big Data builds upon that definition and has the underlying assumption that with such large data sets relationships and correlations are hidden within, which allows describing phenomena in great detail[34;10;6]. In the context of Geo-Information Science (GIS), Geographic Knowledge Discovery (GKD) is similar to KDD with the difference that GKD concerns low dimensional spatial data which is interrelated. The distance measures for GKD are often some form of Euclidean Distance which fits well with our experience of reality, something that is often much less obvious when dealing with distance between n-dimensional data[39;38]. Spatial Dependence, described as "the tendency of attributes at some locations in space to be re-

Table 1: Newtonian approaches versus complex approaches. A Newtonian approach assumes that a phenomenon is predictable and that the outcome is the sum of parts. A complex approach takes unpredictability, uncertainties, adaptability and external factors into account (table adapted from Eoyang 1997[17]).

Use Newtonian approach when the problem is	**Use a complex approach when the problem is**
Quite familiar	New and unique
Well defined	Fuzzy and unknown
Closed to outside influences	Open to outside influences
Small number of people you know well	Related to a large volume of people
One you have tried to solve and succeeded	One you have tried to solve and failed
Linear, the inputs and outputs are clearly distinguishable	Nonlinear, the inputs and outputs are not clearly distinguishable

lated"[39], is one of the building stones for GKD. This concept gives theoretical underpinning for correlating events in space. Though this should be done with care as not all events necessarily are spatially dependent.

For Knowledge Discovery goals can be distinguished between Verification, which tries to verify a hypothesis defined by a user, and Discovery, where the system attempts to find patterns and correlations. Discovery can be subdivided into Prediction, where the system attempts to predict future phenomena by analysis of patterns and correlations, and Description, where the system translates complicated data into information understandable for users[18]. Big Data analysis is especially suitable for describing complex phenomena. Big Data analysis does not strife to find an understanding on why phenomena happen as they happen. The need to simulate complex phenomena becomes less important when the data from those complex phenomena is available for analysis. Big Data analysis allows analysts to describe with large accuracy how these phenomena happen[10].

Knowledge building is a step-wise process. Conventional data knowledge models such as the DIKUW pyramid describe five levels of knowledge: Data, Information, Knowledge, Understanding and Wisdom[45;7;39]. For Big Data analysis, Knowledge, Understanding and Wisdom are not quite as applicable while Data is too broad. The Knowledge Discovery pyramid for Big Data analysis more closely resembles the steps as outlined in the proposed BSID pyramid (Figure 3). The BSID pyramid can be seen as a model outlining the steps for Knowledge Dicovery with the goal to Describe. Describing data does not require Knowledge, Understanding or Wisdom about the data. Also, the step from Big Data is one of where significant value is added to the data. In this research, the BSID pyramid describes the required steps to move from Twitter (Big Data) to Events (Describing).

2.2 Related Works

Conventional Twitter research has been done on a Twitter stream filtered by keywords, which implicates that only a specific interest area is retrieved (e.g.,[42;46]). A downside of this methodology is that a filtered stream is no guarantee for homogeneous data. To homogenize the data, researchers have tried to detect events by using Big Data analysis in support of the already existing structuring mechanisms.

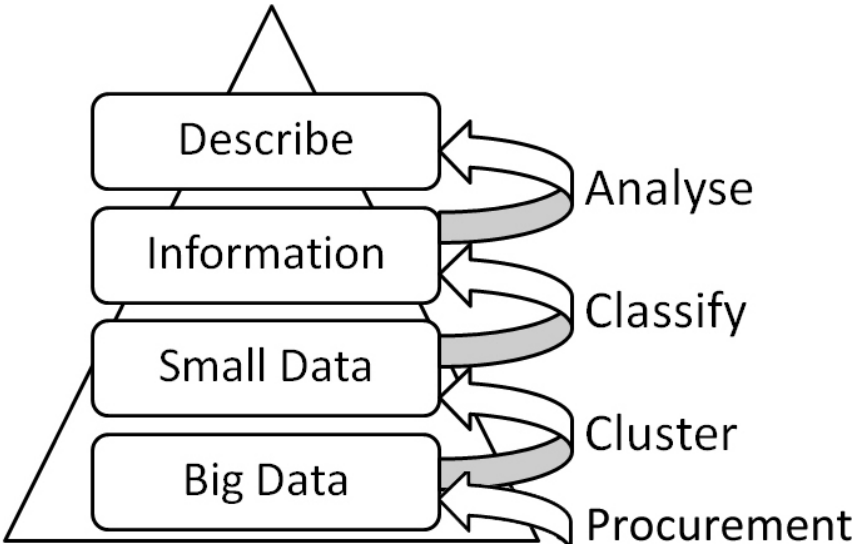

Figure 3: The BSID Pyramid, derived from the DIKUW pyramid but applied to Big Data. At the Big Data level unstructured data in large quantities can be found, such as Tweets, which will need to be procured and stored. At the Small Data level the Big Data has been clustered, generalized and/or associated into several collections. At the Information level the Small Data is classified giving an interpretation of the data. At the Describe level the processes and the relations between processes can be described.

Cataldi et al. [14] have performed event detection on an unfiltered Twitter stream using both the temporal and context dimension of tweets. In this research each word is given a "content energy", which considers the frequency of word usage in a time period. Words that show a similar pattern in usage over a time period are correlated to form events. Similar work has been done by Weng and Lee [54] who analysed Twitter by creating signals for each word and relating this to a volume and time dimension. Bursts of word usage are captured and correlated to other bursts of word usage, the correlated bursts form events. Vossen et al. [53] performed event classification in three dimensions: spatial, temporal and context. Events are identified by finding strong spatio-temporal signals, similar to context/temporal bursts. The dominant word usage within this spatio-temporal collection is used to classify the event. Marcus et al. [35] proposed a methodology that detects and clusters events from Twitter using the context. Their methodology aims to move from Small Data to Information by supervised classification. The downside of this methodology is that it depends supervised classification which is unrealistic in the scope of Big Data. All three approaches consider bursts of similar Tweets as an identifier of an event, usually a sudden increase in Tweets at a time or location (high $\frac{volume}{time}$ or $\frac{spatial}{time}$). A limitation of event detection on Tweet "bursts" is that it requires abnormal signals, events that occur normally or events with a lesser impact or less likely to be identified. Additionally there is an assumption of homogeneity in the event clusters while this might not be the case.

The context, in the case of Tweets often the textual component, is commonly analysed using the $tfidf$ method by Salton and Buckley [47]; Zhao et al. [61]; Cataldi et al. [14] a methodology that puts a weight on terms to select keywords (explored in subsection 3.5). The key words are given a score based on how important the keyword is in a set of observations. Salton and Buckley [47] enhance the $tfidf$ context by spatio temporal analysis of the use of keywords.

Both temporal and spatial data are used in support of the context data, not as a measure on its own. This might limit the type of events that can be detected. Trend detection has been done by Zhao et al.[61] in which social text streams are analysed by considering a context, temporal and social dimension. Using clustering techniques on the context to extract topics and then analysing information flows of topics between actors to detect events. The inclusion of the actor dimension in event detection is an important finding for the event definition. However, not all events have a strong actor/social component which limits the amount of events that can be detected.

Alvanaki et al.[3] have done event detection by considering that hashtags can be correlated to other hashtags, combining these hashtag collections if they are correlated shows an overlapping event. In their research a step is taken from information (hash tag collections) to describing (events). This approach heavily relies on the assumption that the structuring mechanisms of Twitter produce homogeneous event collections. Users might incorrectly label their Tweets, or use the same label while the Tweets concern different events. Zubiaga et al.[62] approached the context dimension differently, not analysing the text words but rather the structure of a Tweet. Indicators of a Tweet are identified such as the presence of exclamation marks, questions marks and website links. These indicators are used to classify a Tweet as: news, current events, memes and commemorative events. The advantage of this methodology is its independence of language as well as relatively easy processing. However similar to[30;27] the classification is not objective which makes it difficult to interpret.

Related works on event detection on Twitter provide key elements which can be built upon. The dimensions of events are a commonly used identifier for event detection. A variety of possibilities on how to detect events using these dimensions is already performed. Analysis of the context dimension is a key component for Event Detection as it is analysed in each of the related works (see the overview of analysed dimensions: Table 2).

Research done on analysing Twitter often considers collections to be homogeneous and complete. No consideration is made that collections consist of sub-collections and Tweets outside the collection might belong to the collection that is analysed. Though Twitter has been analysed using a complex approach (BDA), the complex nature of Twitter is not fully recognized and an underlying paradigm of how to analyse Twitter is missing.
The methodology proposed in this research builds upon the clustering methodologies as proposed in related works[14;54;53;47;61]. The *tfidf* algorithm can be used to enhance the clustering procedure, moving from Small data to Big Data. This research includes the spatial dimension in the event detection methodology as it has barely been explored in related works. Classification of clustered Tweets has been identified as an important component for Event Detection by Marcus et al.[35]; Zubiaga et al.[62]. However the classifications were either arbitrary or supervised. This research proposes an objective automated methodology for classification to cope with the volume and velocity of the Twitter platform. The work from Alvanaki et al.[3] is an example of the last step in the BSID model, used to move from Information to Describing.

Event Definition

In related works the term event is either not defined[3;1] or given an ad hoc definition depending on the goal of the research. Zhao et al.[61] define an event as: "the information flow between a group of social actors on a specific topic over a certain time period." which approaches the event defined by the actor dimension. It does not take into account that events are multi-dimensional in nature. Metzler et al.[37] mention the scalability of events in which they differentiate between different spatial scales. Vossen et al.[53] define events as spatio-temporal-contextual entities but use the context as the main criteria for event detection. Sakaki et al.[46] define the event as "An event is an arbitrary classification of a space/time region" and "An event might have actively participating agents, passive factors, products,

Table 2: Dimensions of an event that have been used to detect events in related works.

Event Detection Research	Event Dimension
Procter et al.(2013)[42]	Context
Sasaki et al.(2013)[46]	Context
Marcus et al.(2013)[35]	Context, Volume/Temporal
Cataldi et al.(2010)[14]	Context, Volume/Temporal
Zhao et al.(2007)[61]	Actors,Context
Weng et al.(2011)[54]	Context, Volume/Temporal
Alvanaki et al.(2012)[3]	Context
Vossen et al.(2009)[53]	Context
Zubiaga et al.(2011)[62]	Context

and a location in space/time." which describes not just agents that subjectively classify the space/time region, but also passive factors which are all the factors used in an event and products which contains everything produced by the event.

Definitions of the term event are numerous and varied. The ad hoc definitions in related works usually reflect what the methodology aims to detect instead of what an event actually is. This makes it difficult to compare research done on event detection as it is unclear what type of events are detected. There is a need for coherent definition of the term event.

3 Methodology

3.1 Event definition

Events can have different meanings depending on the field of research. The goal of this research is to detect events in a similar way as understood by humans. The way that Sakaki et al.[46] defined events ("An event is an arbitrary classification of a space/time region") is an important consideration to make. Events are not absolute and when/where an event ends and begins is highly subjective. Still common ground can be found in the way events are identified and distinguished.

Event Ontology by Raimond and Abdallah[43] defined an event as a multi dimensional entity which has a cause, an effect, a time, a place and agents that experience the event. The ontology also acknowledges a hierarchy in events indicating that an event can be a sub-event of a larger event. This ontology gives a basic summary in what has already been found in related works. However, even though events have a cause and effect it might not be of enough significance to Tweet about. Similarly, agents do not necessarily need to experience an event in order to Tweet about it. Overall, though Tweets can have large similarities to real happenings, clear differences can be observed which should be taken into account for event detection.

In this research a distinction is made between the physical and digital reality events (PR and DR respectively). For PR event detection happens inside the brain; happenings are perceived by a combination of senses, brought together and correlated to form an event[60]. These events can be broken down in sub events which is called event segmentation[29]. Segmentation between events is perceived as such due to change in dimensions[40;60]. Distinctions between (sub-)events are made when a clear difference in dimensionality is observed. Whether the observed event is a sub-event or a new event depends on whether the dimensionality of the event largely stays the same. For example, the spatial dimension changes but the context stays the same. In DR, data is in a format that is not easily processed as PR in the brain. It is nearly impossible to read every Tweet, which can be seen as a happening. Still, a similar approach can be taken for event detection. Using computer analysis Tweets can be clustered based on similarity in dimensions to form events and distinguish between events.

PR events are classified in the brain (see Figure 4). For example "a wedding" is a classification of an event with an expectation of context (getting married, music, rings etc.), actors (bride, groom, best man, family and friends), duration (hours) and location (church or municipality)[60]. In the DR, clusters of similar happenings can also be classified using a centroid, a volume and the amount of clustering. The classification gives a measurable objective interpretation of clusters. Additionally it serves as an identifier to which other events it might be related.

For this research a combination of definitions is proposed. A graphical representation of the event definition is shown in Figure 5, in which an event is seen as an entity of multiple dimensions that have quantifiable properties.

The definition describes the event as an entity, that is a collection of happenings. An event can be or contain sub-events which in turn are a collection of happenings. Events are an arbitrary classification of happenings that show similarities in dimensions. This is in line with the findings of Kurby and Zacks[29] and Zacks et al.[60] and the definition proposed by Sakaki et al.[46].

Based on related works as shown in Table 2 and the works of Raimond and Abdallah[43] 4 dimensions are distinguished. The *context dimension* indicates what happened, cause, effect and the actual event. This dimension is a combination of perceptions. In the case of Twitter (DR), PR perceptions such as sight, taste, smell etc. are less profound and captured in the

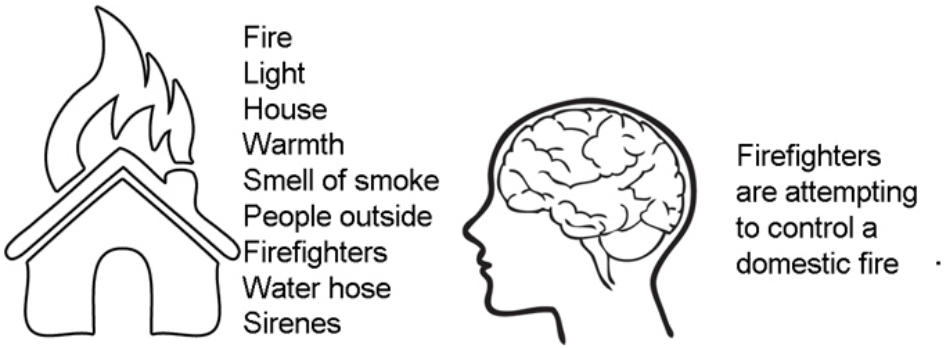

Figure 4: Event detection in the PR. A combination of senses is used for perception of happenings. The happenings are clustered to identify and classify events.

text. For Twitter it consists of the text of the Tweet, when texts within an event are strongly clustered the event is specific and vice versa broad. The amount of clustering in the context dimension gives an indication about the broadness of an event. For example, a sunny day can be an event were people go to the beach, participate in sports, and drinking a cocktail on the terrace. An example of an event with a broad context. The *spatial dimension* indicates the place where an event takes place. It can show correlations and clustering in space. An event with a low/high amount of spatial clustering can be classified as global or local respectively, which can be important information about the nature of the event. The *temporal dimension* indicates when something happened and the duration of an event. An event clustered in the temporal dimension is the difference between a constant stream of happenings that constitute to one event or the same amount of happenings at a certain time. The *agent dimension* indicates who is experiencing the event, whether this is a group of people or individuals. The amount of clustering in the agent dimension gives an indication on the importance of specific agents in the context of an event. For example a concert is an event that is largely centralized around a small group of agents (the band) though experienced by many. An event like New Year's Eve is experienced by many, but the event is not centralized around specific agents.

Dimensions can be characterized using properties. The *centroid* property is the average of the entire dimension, often expressed as a vector. Centroids can be used as a quick identifier to recognize similar events. The *volume* property indicates the extent of a dimension.[36] mentioned the scalability of events in the spatial dimension. However, scalability can be found in every dimension: the amount of words that occur in a Tweet, the spatial extent of the event, the duration of the event and the amount of agents that experience the event. Similarities in volume combined with another dimension can be used to correlate events, for instance the space/volume and time/volume of a recurring celebration is likely correlated over the years. Though a property of a dimension, a volume can also be used to give an indication of the impact of an event. Counting the amount of happenings that constitute to an event (in this research, the amount of tweets). This characteristic has been used in Event Detection research using the "bursts" by Cataldi et al.[14]; Vossen et al.[53]; Weng and Lee[54]. The amount of *clustering* within a collection gives information on the relative importance of a dimension within an event. For example: events tightly clustered in space indicate that the location of an event is of importance for the event.

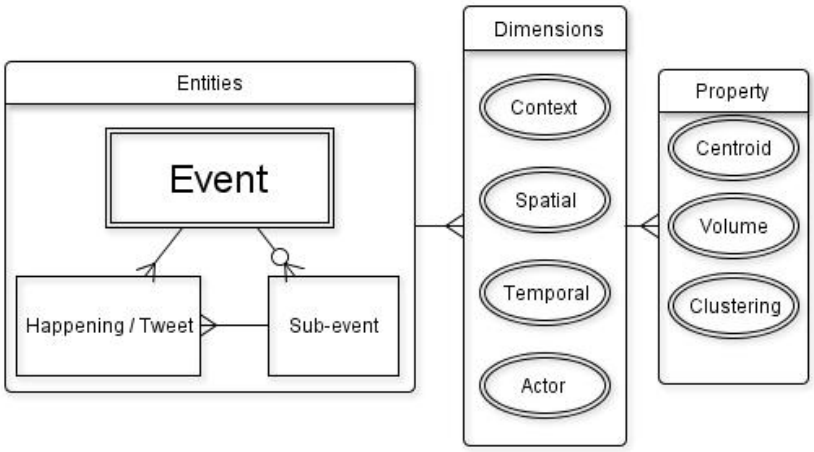

Figure 5: Proposed DR "event" definition illustrated as a multi-dimensional entity. Events are built up from happenings and can contain sub-events. The *context*, *spatial*, *temporal* and *actor* dimension describe an event. Each of these dimensions can be quantified in properties. The properties are a *centroid*, which is the average value of a dimension, a *volume*, which indicates the extent of the dimension, and *clustering*, which is a measure for the amount of clustering within an event.

3.2 Conceptual Model

Methodology
The goal of this research is to create a methodology that allows event detection on Twitter in any dimension. In this paper the focus will lie on only two dimensions but the same principles can be applied for other dimensions. In abstract, the methodology follows the steps as shown in the BSID pyramid Figure 3. In item (ii) and item (iii) the requirements and assumptions respectively of the methodology are outlined.

A happening is the smallest entity/classification in dimensions. It is a building block for collections, clusters and events. In this research a happening is a single Tweet, in line with Figure 5. A collection (C) is an agglomerate of happenings. Tweets within the collection do not necessarily need to be related in dimensions, though they can be. In terms of knowledge level, the collection is Big Data. A cluster (c) is an agglomerate of happenings within a collection ($c \in C$) which have been put together due to similarities in dimensions. In terms of knowledge level the clusters are Small Data. An event is a classified cluster of happenings. Events detected in the DR are not necessarily recognized as events in the PR. An event cluster with a large amount of clustering in the actor and spatial dimension is not comparable to events that the brain would recognize.

Event detection starts with a collection as input. This can be the entire database of Tweets or a selection on the database to reduce processing time. This collection is clustered based on similarities between dimensions (item (iii), 4 and 6). Similarities in dimensions can be identified using distance or similarity measures (item (iii) point 3). In the process of clustering the ideal amount of clusters and the ideal cluster distribution is calculated. The optimal solution is classified based on the centroid, volume and clustering for each cluster in each dimension (see Figure 6). As a result the collection is divided into classified events. A schematic overview of the implementation can be found in Figure 7.

The clusters can be classified using the centroids, volume and clustering for each dimension.

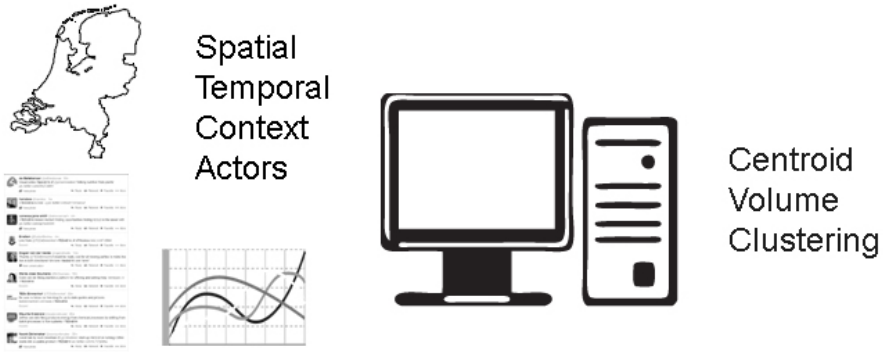

Figure 6: Event detection in the DR. Each Tweet has characteristics which can be divided in four dimensions: context, spatial, temporal and actor. Tweets are clustered based on similarity in these dimensions which gives a classification that includes a centroid, volume and clustering in each dimension.

While this classification is less convenient for use than for instance "Christmas" or "Lunch break", it is an objective repeatable classification that allows linking clusters together based on these attributes. The meaning of these attributes differs greatly depending on the topic analysed. A cluster with a spatial extent of 40 km (volume, spatial) might be large for a music festival but small for weather phenomena. As such, the classification is not translated into language as is convention for PR events. However, for a computer an event that takes place on December 25th (centroid, temporal), with the duration of a day (volume, temporal) with a large spatial extent (volume, spatial) and no real spatial clustering (clustering, spatial) with "christmas, presents, turkey, etc." as most occurring words (centroid, context) and not centred around specific actors (clustering, actors) is easily recognizable as being related. In essence, this classification is similar to the translation that is made inside the brain when labelling an event as explored by Zacks et al. [60].

(ii) **Requirements**

1. *Polymorphism* - the methodology needs to be capable of detecting clusters in multiple dimensions using the same building blocks.
2. The methodology needs to be capable to go from small data to information (BSID pyramid Figure 3).
3. The methodology needs to be capable of identifying clusters in multiple dimensions in an objective manner repeatable.
4. The methodology needs to be capable of detecting clusters within collections and differentiating between events.
5. The methodology needs to be capable of classifying clusters using the properties of the dimensions as proposed in Figure 5.

(iii) **Assumptions**

1. Twitter is a CAS, which should be analysed using complex methodologies. BDA is a suitable method for event detection in a CAS (subsection 2.1).

2. Events are multi-dimensional entities with four dimensions: agent, spatial context and temporal dimensions. The dimensions can be described using properties: centroid, volume and clustering (subsection 3.1).
3. A low distance in dimensions between happenings indicates similarity.
4. Events can be identified by clustering happenings that share similarities in dimensions (subsection 3.1).
5. Separation between events occurs when there is a change in one or more dimensions (subsection 3.1).
6. Spatial Dependence as described by Miller and Han[39] can be an identifier for event detection.

Restrictions
In this research only part of the methodology is implemented. The choice is made to only consider the textual and spatial dimension due to time constraints. The spatial dimension is chosen as it has barely been explored in related works (see Table 2). The context dimension is chosen due to its importance for event detection is it encompasses many dimensions. Without the context it is difficult to interpret events.

The relation between DR and PR is not explored in this research. The strength of this relationship likely depends on the topic. This means that for each topic of interest a parameter study would need to be done. Though it is of interest whether Twitter events can serve as a model for events in the PR, a large value can also be gained by considering DR events on its own. Much of the BDA research can create value by only considering the DR. The methodology as proposed in this research can serve either goal depending on the interest.

The final step of Describing the data (following the BSID knowledge pyramid) is not made in this research. A large potential lies in relating events together revealing patterns in events as well as drawing relations between events. This is a similar step as moving from Big Data to Small Data, the amount of data to analyse decreases creating higher value data. Describing the classified clusters in the dataset can be done by clustering events together similar to the work of Alvanaki et al.[3]. However, due to time constraints it could not be realized within the boundaries of this research.

3.3 Data Procurement

The procurement of Tweets (source data) can be done using the Twitter REST API, an Applied Programming Interface that streams data. The REST API stream gives a small fraction of the total number of messages sent at any time. Using the tools of the Twitter API, filters can be applied for words, locations users and message types (replies, retweets). The Twitter stream can be connected by using several programming language interfaces such as Java and Python. For each of these platforms a multitude of modules are available which can be tweaked to reflect the requirements of this project.

The Tweepy module available for the Python programming language allows the initiation of a filtered Tweet stream that returns a JSON object the content of the Tweet, the username, the geotagged location and the time Code 1.

```
1 #ENABLE TWEETSTREAM
2 #import the neccesary module
3 import tweepy
4 login= consumer_key, consumer_secret, acces_token, acces_token_secret
5 tweetstream = authenticate(login)
6 #filter the tweetstream based on a square of longitudes and latitudes
7 filterstream = tweetstream (locations: lon , lat, lon, lat )
```

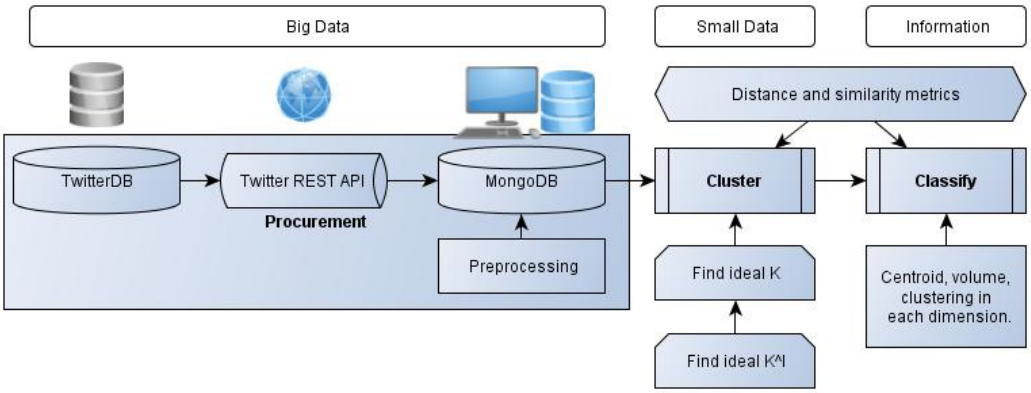

Figure 7: Schematic overview of the implementation of the BSID model for Event Detection. From Big Data to Information; procurement (subsection 3.3), preprocessing (subsection 3.4), clustering (subsection 3.6) and classification (subsection 3.7). Clustering and classification require distance metrics that are examined in subsection 3.5

.

```
#write detected JSON object to a database if it contains a geotag
if filterstream detects JSONobject:
  #geotag can be either a geo reference in coordinates or a placename.
  if tweetobject contains geotag:
    writeToDataBase(JSONobject)
```

Code 1: Basic procedure in pseudocode for the procurement of Tweets using the Tweepy module

The tweepy module downloads a JSON object, similar to a dictionary, which contains information on a wide spectrum of aspects of a Tweet which can be used for analysis, documentation can be found[52]. For the purpose of this research, a selection of attributes has been made on which the analysis is to be performed. A full overview of the attributes stored in the database can be found in Appendix A, Table 6.

The data from the Twitter feed is stored inside a Mongo database which is chosen for its flexibility and intuitive handling of queries inside Python. Accessing this database is done with the pymongo module which enables a connection to databases and collections inside databases as seen in Code 2.

```
import pymongo
class mongoInt(object):
    def pymongoconn(self,mdb='default'):
        try:
            # Initialize an object to connect to the MongoDB
            conn=pymongo.MongoClient()
            # Make the connection to the database where mdb refers to the name
                of the database
            self.db = conn[mdb]
            # Connect to the collection inside the database
            self.tc = self.db.tweets
        except pymongo.errors.ConnectionFailure, e:
            print "Could not connect to MongoDB: %s" % e
```

Code 2: Basic procedure in Python for accessing MongoDB

As Big Data inherently requires a large amount of computing time it is of great importance that analysis of Big Data is done in an efficient matter. For this program a standard format for queries is used which is both the result and input for all analysis. The advantage of using queries as input and output of analysis is that a database is much more efficient at selecting parts of data than the analysis is. Additionally, results can be saved without putting all entries in memory which drastically speeds up processing. The query construct which from now on will be referred to as a collection ($col \in T$), is generated by using the standard dictionary format of Python using Code 3. When the collection is used as an input for MongoDB it will read the dictionary as a query that is translated to SQL as shown in Code 4.

```
#Function call, requires a list of IDs as input
def convertToCol(list=[]):
    #Initiate a dictionary
    col = {}
    #Add to the dictionary the list with the the query
    col['id'] = {'$in':list}
    return col
```

Code 3: Procedure in Python for converting lists to queries

```
SELECT *
FROM MongoDB
WHERE 'id' IN list
```

Code 4: Translation of the collection query in SQL

3.4 Preprocessing

The text of a Tweet can contain components that are useful for classification of the Tweets. These components are extracted in using an automated methodology called Regular Expression, an optimized methodology for scanning large volumes of text. As an example: websites, hashtags, mentions, retweets, (place)names and identifiers for mood can be extracted using this methodology.

Similarities between the context (text) of a Tweet can also serve as a measure for classification and clustering. Stop words in the text are removed using the stop word collection from the Natural Language Toolkit (NLTK)[8]. This is done for two reasons: 1. Similarity in stop words does not necessarily indicate a common topic between Tweets, 2. To speed up processing. The text without stop words object is vectorized(iv) for use in Cosine Similarity calculations.

(iv)
1. Remove stop words using the NLTK.
2. Split the text in words, space delimited.
3. Remove all non-letter characters.
4. Count the occurrence of words.
5. Represent the text as a weighted vector "Hello Amsterdam! Life is good #amsterdam" becomes {hello:1, amsterdam:2, life:1, good:1}.

The Twitter REST API contains coordinates as part of the JSON-object if the user permits the Tweet to send the location. If no location is available a Null value is put in the JSON object (see Table 6). An alternative way to retrieve the coordinates of an object is by interpreting the toponym location which is stored in the *place.name* of the API using a

geocoder. For this research, the Google geo-coder V3 is used[20] which can be queried 25000 times/day, each query is stored in a database after each successful query. The *place.name* can be as specific as the user requires, it is either manually filled in, or specified by the user by doing a query based on the coordinates, the REST API return a list of suggestions which correspond with the coordinates in several scales. This means that the place object can also contain data on what the subject is about instead of where the user is situated, however, which one is the case is not specified[52]. Potentially, this source of information is more valuable than the coordinates of a Tweet, but it also has the potential to contain erroneous or less specific information. The *capital* attribute (see Table 6) can also be used to deduct a location from the Tweet. For each word a check is done whether it is a geo-location indicator using the geo-coder. For this research the *captial* and *place.name* are only used when the *coordinates* attribute is not available.

Coordinates are converted from decimal to radians for use with the Haversine formula (Equation 1).

$$coords_{radians} = \frac{coords_{decimal} * math.pi * 2}{360} \tag{1}$$

3.5 Distance and similarity metrics

Spatial distance between coordinates is calculated using Equation 2,3 and 4. The Haversine formula calculates the distance between two positions lat/lon in decimal units, the output is the distance in kilometre[56].

$$a^2 = \sin(\frac{lat_2 - lat_1}{2}) + \cos(lat_1) \cdot \cos(lat_2) \cdot \sin(\frac{lon_2 - lon_1}{2})^2 \tag{2}$$

$$b^2 = 1 - a^2 \tag{3}$$

$$distance = (2 \cdot \arctan(\frac{a}{b})) * 6367 \tag{4}$$

For text analysis, methods for text similarity can be grouped into lexical and probabilistic. Lexical models directly compare vectors of text against each other while a probabilistic model creates a probability distribution to cluster text[50] also a combination can be used providing better results at the cost of computing time[36]. As the focus of this research does not lie in optimizing textual clusters a lexical model is used due to its speed. Because Tweets are a sparse type of data ($words \notin Tweet \gg words \in Tweet$) a lexical model is chosen that only takes non zero values into account: Cosine Similarity (CS), a statistical measure that indicates how similar two collections of vectors are as well as a distance metric that shows similarity between individual vectors[44]. CS is especially useful for Twitter text analysis as it normalizes document length, which has a large variation amongst Tweets[16]. The formula for calculating the CS can be found in Equation 5:

$$\cos(x, y) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| * |\vec{y}|} \tag{5}$$

In which $x \cdot y$ is the dot product ($\sum_{i=1}^{N} x_i * y_i$) and x is the length of the vector x ($\sqrt{\sum_{i=1}^{N} x_i^2}$). CS values can range between 0 and 1 which indicates no and perfect correspondence respectively[55;48]. Usually values can range between -1 and 1 but negative values are not possible for text applications.

A downside of CS is that homonyms are not taken into account. Additionally, all words

are deemed equally important using the CS, this does not reflect the actual importance of words in a Tweet. Another aspect of CS is that a sparse data volume puts a much larger emphasis on a single mismatch. As such it works better for comparing texts than hashtags and mentions (# or @). Comparing short Tweets will more likely give more extreme cosine similarity values scores than comparing long Tweets. These downsides are common amongst lexical analysis of text[36] and other lexical small text clustering algorithms do not necessarily perform better[48]. Methodologies to mitigate these issues have been suggested, putting more or less emphasis on either the missing, overlapping or extra terms[16]. But also normalizing the weight of each word inside the vector by creating a vector space model known as the $termfrequency$ (tf) $inversedocumentfrequency$ (idf) model, which diminishes the weight of words that appear in many Tweets[44;47;61;14]. The $tf(t,T)$ factor is calculated by frequency f of a term t in a Tweet T (Equation 6). The idf factor is calculated by taking the total number of Tweets N and dividing that by the number of times a term t occurs in a Tweet T that is part of the database D. The tf is multiplied with the idf term giving a weight $w(t,D)$ (Equation 8) which can be substituted in Equation 5 as shown in Equation 9. The full code for the calculation of the *tfidf* can be found in Appendix G, Code 9.

$$tf(t,T) = f(t,T) \tag{6}$$

$$idf(t,D) = \ln(\frac{N}{\{t \in T, T \in D\}}) \tag{7}$$

$$wt,d = tf(t,T) \times idf(t,D) \tag{8}$$

$$\cos(x,y) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| * |\vec{y}|} = \frac{\sum_{i=1}^{N} w_{i,x} \times w_{i,y}}{\sqrt{\sum_{i=1}^{N} w_{i,x}^2} \times \sqrt{\sum_{i=1}^{N} w_{i,y}}} \tag{9}$$

3.6 Clustering

The *K-Means* algorithm is a common clustering methodology characterized by its simplicity and ease of implementation and efficiency in handling large datasets. It is used in data mining, text analysis[24;26;44] and typical GIS applications[4;13] *K-Means* requires four user inputs: the number of clusters (K), a cut-off value(c), the initial cluster centroids C and a distance metric. It is an iterative method described in item (v) and Figure 8.

(v)
1. Choose a value for K and c.
2. Assign initial centroids C.
3. Calculate distance from point (vector/spatial) to centroid $d(C,p)$.
4. Assign each point to a cluster from which the centroid is closest to the point.
5. Recalculate the centroid of the cluster.
6. Repeat steps 3-5.

The most suitable value for K is chosen by running the algorithm independently for different K values[24]. Still the choice for an optimal K value is ambiguous, as increasing the K value will always decrease the error until the error is zero when $K = N$. Usually, the percentage of variance explained (PVE) as a function of the number clusters will show an elbow shape as shown in Figure 9 indicating an ideal solution for the amount of clusters K[57]. The PVE is calculated by taking the ratio of the Between Sum of Squares BSS) and

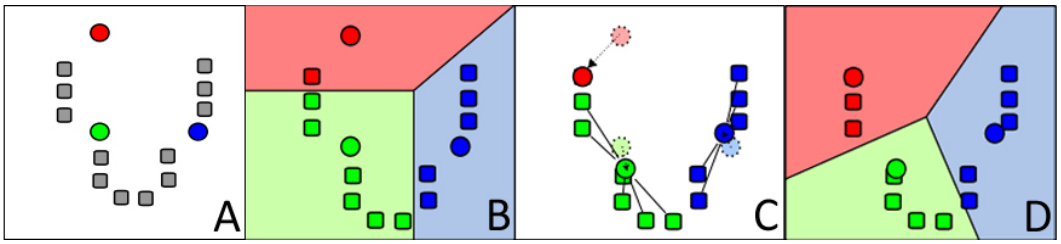

Figure 8: *K-Means* algorithm in steps. (A) Centroids, shown as circles, of the clusters are randomly chosen. (B) The points, shown as squares, are added to a cluster based on the distance to the centroid. (C) The update step: new centroids are calculated and the centroid shifts to a new mean. (D) Step 2 and 3 are repeated until the centroid shift is smaller than the user defined threshold c. Images adapted from [57].

the Total Sum of Squares (TSS). These are calculated using the Within Sum of Squares (WSS) as seen in Equation 10:

$$WSS = \sum_{i=1}^{N} d(C, p_i)^2 \text{ if } K > 1 \tag{10}$$

$$TSS = \sum_{i=1}^{N} d(C, p_i)^2 \text{ if } K = 1 \tag{11}$$

$$BSS = TSS - WSS \tag{12}$$

$$PVE = \frac{BSS}{TSS} \tag{13}$$

In which $d(C, p)$ is the distance from each point that belongs to a cluster to its centroid. The TSS is calculated by using the formula for the WSS with only one centroid (Equation 11). The BSS is calculated by detracting the WSS from the TSS (Equation 12). The full code for the *K-Means* algorithm can be found in Appendix H, Code 10.

The initial cluster centroids are picked randomly from the existing collection, making sure that the distance between centroids is larger than the cut-off value ($d(C_x, C_y) > c$). *K-Means* as an iterative technique, is especially sensitive to starting conditions. It converges to a local minima which can lead to unintuitive results as shown in Figure 10 [24]. A common method to overcome the local minima, is to optimize the initial cluster centroids by repeating the *K-Means* algorithm multiple times and choosing the ideal initial centroid based on the PVE value, the amount of initiations done is labelled as $K^{I\,24;11}$. In the update step, the centroid is recalculated (item (v) step 4) by calculating the mean of all points as shown in Equation 14:

$$C^{t+1} = \frac{1}{N} \sum_{i=1}^{N} p_i \tag{14}$$

In which C^{t+1} is the calculated centroid for the next iteration, N the total number of points inside a cluster and p_i point i in a collection of points ($p_i \in col(P)$). The algorithm for finding the ideal centroid using the calculation for the PVE can be found in Appendix I, Code 12.

The distance metric for text clustering is the distance between vectors for which in this research the CS is used (Equation 5). For clustering in the spatial dimension the Haversine formula (Equation 3) is used as a distance metric

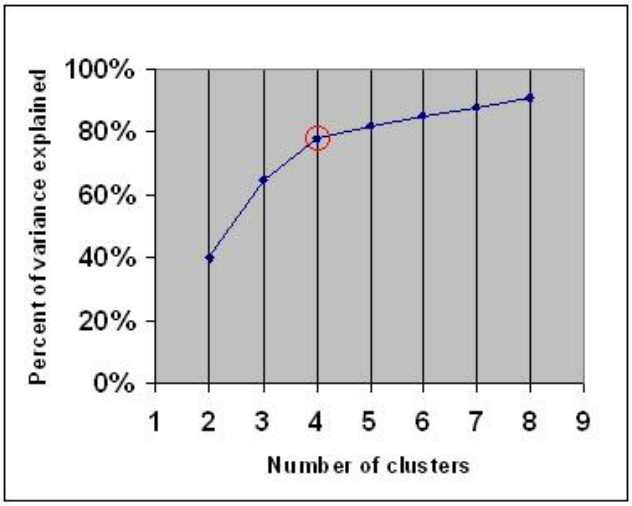

Figure 9: In data set clustering, the PVE value increases with the amount of clusters. However, the first clusters explain a larger amount of the TSS than later clusters. The point where the the PVE flattens when adding subsequent clusters is referred to as the elbow point, which is seen as the optimal amount of clusters. In this example at 4 clusters (image adapted from[5]).

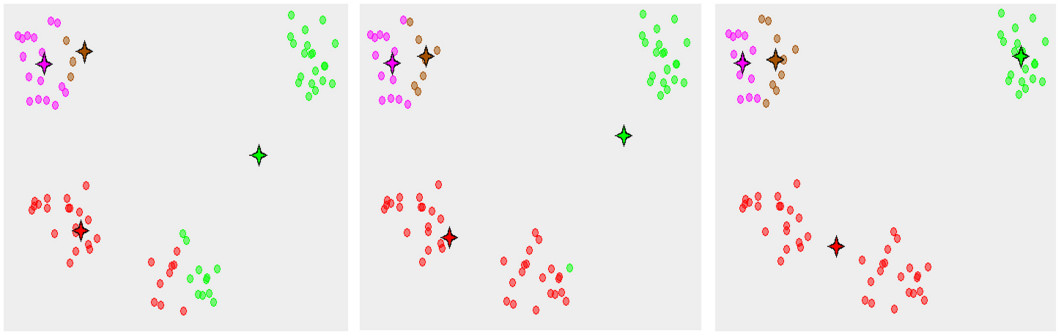

Figure 10: A typical result for *K-Means* clustering where the clusters converge to a local minima, giving unintuitive results (image adapted from[41]).

3.7 Classification

As explored in subsection 3.2, dimensions of an event can be described with three properties: *centroid*, *volume* and *clustering* (item (vi)):

(vi)
1. *Centroids*, the centroids of a collection in each dimension.
2. *Volume*, the extent of a dimension.
3. *Amount of clustering*, a value that gives an indication on the amount of clustering in a dimension.

From which the *centroid* can be calculated using Equation 14. The *volume* can be calculated by calculating the extent (temporal,spatial) or the amount (amount of agents, amount of words used). As examined in subsection 3.1 the number of Tweets can also be a characteristic used to classify an event. A measure for *clustering* in the spatial dimension is the k Nearest Neighbour (kNN) algorithm, an algorithm that is commonly used in data mining[59] and spatial analysis software such as ESRI ArcGIS 10.1. This algorithm can be used to classify the amount of clustering inside Tweet collections as well as finding k nearest Tweets. The kNN value calculated ranges from 0-∞ in which a lower value indicates a larger clustering within the dataset. Equation 17 shows the basic formula for calculating the kNN:

$$D_O = \frac{\sum_{i=1}^{k} d_i}{k} \tag{15}$$

$$D_E = \frac{0.5}{\sqrt{n/A}} \tag{16}$$

$$kNN = \frac{\overline{D}_O}{\overline{D}_E} \tag{17}$$

In which the D_o is the sum of the observed mean distances to k nearest neighbours d_i divided by k as shown in Equation 15. The D_e is the expected mean distance which is calculated using the total number of features n and the overall area of the study area A as shown in Equation 16. The interpretation of the kNN value depends on the value chosen for k which in turn, depends on the purpose of the analysis and the dataset. When classifying a feature often a lower k value will suffice. Larger k values reduce the effect of noise on the classification, while low k can lead to a high kNN standard deviation ($kNNsd$) within the collection. Due to collections varying in size, a fixed k will influence the classification. As such, the average nearest neighbour (when $k = N, kNN = \overline{NN}$) is used for classifying collections. The full code for the *kNN* can be found in Appendix F, Code 7.

When classifying the *clustering* in the context dimension, the CS is used as a distance metric in a similar way as the Haversine formula is used in kNN for spatial applications. The kCS value is calculated by taking the average CS value of k nearest neighbours. The interpretation of the value is opposite however, the the scale for kCS is reversed (0 to 1) compared to kNN (∞ to 0). The full code for the *kCS* can be found in Appendix G, Code 8.

4 Results and Discussion

4.1 Procedure

To evaluate the performance of the methodology and implementation as proposed in this research, several case studies are designed. The requirements as outlined in item (ii) point 3, 4 and 5: *Differentiation between events*, *Within collection clustering*, *Cluster in multiple dimensions* and *Classify in multiple dimensions*. Finally an overall discussion on the strengths and weaknesses of the proposed methodology is done.

The Twitter data used is captured on the 5th of December 2013 between 12:00 and 19:00. The Twitter stream is filtered, receiving only Tweets with (place_)coordinates between Lon 3.22-7.22 and Lat 50.75-53.33. giving a total of $\approx$ 51.000 Tweets.

Differences between collections are measured using Cohen's Kappa statistics (Equation 18 [58]).

$$\kappa = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)} \tag{18}$$

In which $\Pr(a)$ is observed percentage of agreement and $\Pr(e)$ the probability of chance agreement. A κ value of 1 indicates perfect agreement and a value of 0 indicates no agreement.

4.2 Differentiation between events

A requirement of the methodology is that it can differentiate between events. The methodology was tested in this regard by performing contextual *K-Means* clustering on collections that consist of two vastly different hash tag collections Table 3 (C1-C3). A collection built up from Tweets is not an event by definition. However, the purpose of this case study is to investigate whether the clustering methodology can differentiate between two obviously different collections. The clusters as a result of context *K-Means* clustering and the original collections are compared. The κ value is calculated, which shows a perfect correspondence (κ of 1.00). This is to be expected as the Tweets of either collection do not have many words in common. In this case it means that the best clustering solution corresponds with the structure provided by Twitter: the hashtags. However, this is not always the case. When comparing two similar collections such as *#onweer* and *#storm* (both translate as storm in English) the clustering does not show perfect correspondence with the hashtag collections (Table 3) (C4,C5). Tweets categorized as *#storm* by users but textually (given a lexical model) 60 Tweets are closer to the *#onweer* collection. A similar phenomena occurs when comparing the similar collections *#pakjesavond* and *#sinterklaas* ("presents evening" and a Dutch celebration in which gifts are given respectively). When Tweets without a hash tag are also considered and more collections are taken into account the classification would change even more. A downside of doing this analysis is that it costs computing time. Additionally, as a lexical model is used homonyms are not taken into account. However, using this methodology does give a more focused homogeneous collection compared to the source data.

4.3 Within collection clustering

One of the requirements of the methodology is that it should be capable of detecting clusters within collections. In line with the definition of the event given in Figure 5. To test this, a collection is built up with from Tweets labelled with #hagel (#hail in English). In Table 4 the results are shown for context clustering. The *kCS* values increase with the amount of clusters indicating that the amount of clustering in the text increases. Also the spatial

Table 3: K-Means context clustering ($K = 2, K = 20$) on five combined collections. Collection 1 (top row, left table) is built up from Tweets with *#ns* and *#peopleschoice* containing 315 Tweets. Collection 2 (top row, middle table) is built up from Tweets of *#hagel* and *#fyra* containing 179 Tweets. Collection 3 (top row, right table) is built up from *#sinterklaas* and *#acakfilm* containing 315 Tweets. Collections 4 (bottom row, left table) is built up from *#onweer* and *#storm* containing 578 Tweets. Collections 5 (bottom row, right table) is built up from *#pakjesavond* and *#sinterklaas* containing 351 Tweets. Intersections between collections are removed. The two resulting clusters are annotated with *C#A* and *C#B*. The original collections are compared to the clusters using Cohen's Kappa statistics. The K-Means clusters for C1, C2 and C3 show a perfect inter-rater-agreement, with a κ value of 1.00. While the κ value for C4=0.42 and C5=0.84. The top 15 words in each cluster can be found in Appendix C.

C1	*C1A*	*C1B*
#ns	100	0
#peopleschoice	0	79

C2	*C2A*	*C2B*
#hagel	53	0
#fyra	0	113

C3	*C3A*	*C3B*
#sinterklaas	239	0
#acakfilm	0	76

C4	*C4A*	*C4B*
#onweer	25	0
#storm	60	493

C5	*C5A*	*C5B*
#pakjesavond	112	0
#sinterklaas	26	213

clustering improves as the *kNN* values decrease with increasing *K*. An overall decreasing standard deviation with increasing *K* values is found. This is partly due to more homogeneous clusters but also due to less Tweets in a collection.

Cluster CTK2C1 is an anomaly that deserves attention as it has a high *kCS* value and a *kNN* value of zero. This makes it likely that the Tweets from the cluster originate from a bot sending out similar messages from one location, which is also the case. That bots can be filtered out is an unforeseen advantage of this methodology.

As observed in Figure 11 the optimal clustering of this collection can be found at two clusters, giving Text collection 1, where the gain in *PVE* is the largest. Optimal in this context is according to the elbow method described in subsection 3.6 which indicates an optimum *PVE/K* ratio. CTK2C1 where *K=2* (corresponds with CTK3C0,CTK4C0,CTK5C0) has not been clustered further with increasing *K* values. In this case because larger differences could be found in the other collection. However, there is a possibility that this cluster can be optimized with further clustering.

Text collection 1: Top 15 most used words in collection as a result of K-Means ($K = 2, K^I = 20, k = N$) clustering of collection #hagel. The clusters correspond with the clusters in Table 4.

Cluster: CTK2C0
{ hagel: 22, windstoten: 22, zie: 22, regen: 22, onweer: 22, storm: 22, nederland: 22, natte: 22, mm: 22, apeldoorn: 22, http://tco/xzm1bmcgdp: 22, sneeuw: 22, coderood: 12, achtig: 10, codegeel: 10 }
Cluster: CTK2C1
{ hagel: 31, onweer: 21, storm: 16, regen: 8, weer: 6, lekker: 3, zware: 2, windvlagen: 2, tekeer: 2, zitten: 2, noodweer: 2, echt: 2, gaat: 2, midden: 2, zeer: 2 }

Table 4: *K-Means* context clustering results ($K^I = 20, k = N$). K is the amount of clusters, *PVE* is the percentage of variance explained, the cluster refers to the cluster as a result of *K-Means* optimization, *kNN(sd)* is the k-nearest neighbour statistic and its standard deviation, *kCS(sd)* is the k-nearest cosine similarity statistic and its standard deviation, N is the amount of Tweets in a collection.

K	PVE	Cluster	kNN	kNNsd	kCS	kCSsd	N
1	0	$HAGEL$	2.966	1.288	0.256	0.115	53
2	0.358	$CTK2C0$	3.141	0.81	0.183	0.073	31
		$CTK2C1$	0	0	0.702	0.012	22
3	0.449	$CTK3C0$	0	0	0.702	0.012	22
		$CTK3C1$	2.6	0.637	0.202	0.074	23
		$CTK3C2$	1.87	0.493	0.188	0.078	8
4	0.507	$CTK4C0$	0	0	0.702	0.012	22
		$CTK4C1$	1.083	0	0.23	0	2
		$CTK4C2$	2.302	0.605	0.222	0.078	20
		$CTK4C3$	2.145	0.492	0.185	0.075	9
5	0.536	$CTK5C0$	0	0	0.702	0.012	22
		$CTK5C1$	2.27	0.586	0.232	0.078	19
		$CTK5C2$	1.083	0	0.23	0	2
		$CTK5C3$	2.145	0.492	0.185	0.075	9
		$CTK5C4$	-	-	-	-	1

When collection CTK2C0 is subjected to *K-Means* clustering an optimal solution is found at *K=2* (Figure 12). As observed in Text collection 2, the clusters are distinguished by the presence of the word "codegeel" in CTK2C1-A (code yellow) and "coderood" in CTK2C1-B (code red), both are indicators for extreme weather. A human interpretation of this distinction could be that the overall event was "extreme weather" with sub events "codegeel" weather and "coderood" weather. This result in which a cluster consists of sub-clusters is in support of the definition of an event given in Figure 5.

Text collection 2: Top 15 most used words in a collection as a result of K-Means ($K = 2, K^I = 20, k = N$)clustering of collection CTK2C1.

Cluster: CTK2C1-A
{ hagel: 12, coderood: 12, windstoten: 12, zie: 12, regen: 12, onweer: 12, storm: 12, nederland: 12, natte: 12, mm: 12, apeldoorn: 12, http://tco/xzm1bmcgdp: 12, sneeuw: 12, 5,1: 3, 10,8: 1 }
Cluster: CTK2C1-B
{ achtig: 10, hagel: 10, codegeel: 10, windstoten: 10, zie: 10, regen: 10, onweer: 10, storm: 10, nederland: 10, natte: 10, mm: 10, apeldoorn: 10, http://tco/xzm1bmcgdp: 10, sneeuw: 10, 14,4: 5 }

4.4 Cluster in multiple dimensions

The methodology needs to be capable of clustering of collections in multiple dimensions. The implementation for clustering the temporal and actor dimension lied beyond the scope of this research. However, an implementation for the spatial dimension is made. Spatial

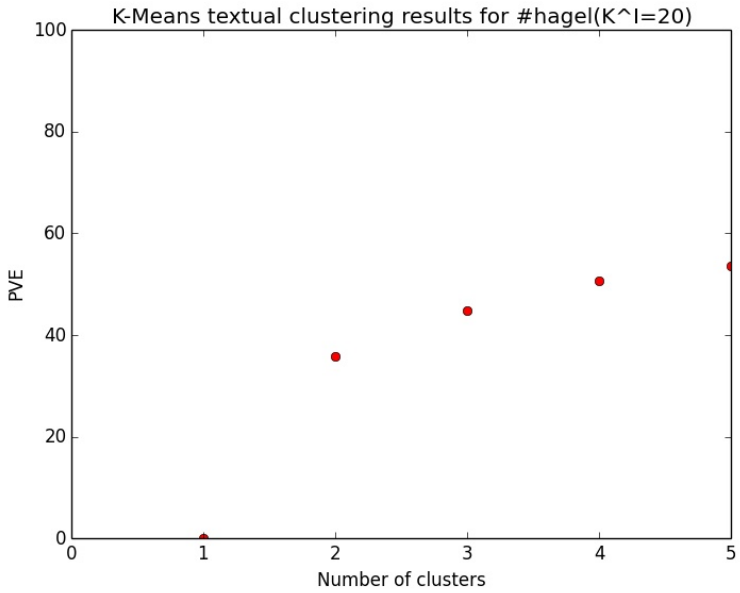

Figure 11: Elbow plot of the data in Table 4 the amount of clusters on the X axis and the PVE value on the Y axis. Most variance is explained when $K = 2$.

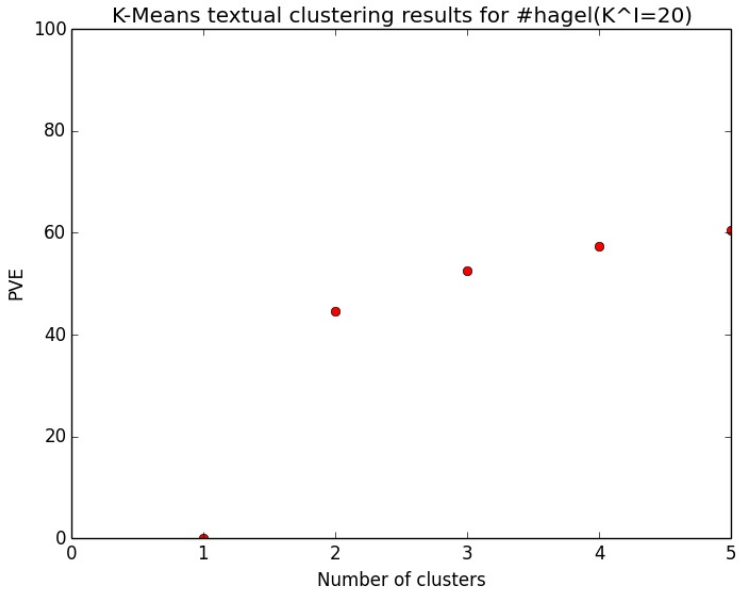

Figure 12: Elbow plot of the data in Table 7 (Appendix B) the amount of clusters on the X axis and the PVE value on the Y axis. Most variance is explained when $K = 2$.

K-Means clustering has been performed on the #hagel collection. The results can be found in Table 5. Overall results show a dropping kNN value with increasing K values, indicating stronger clustering. The $kNNsd$ values also decrease which indicates more homogenized collections. The weighted average of kCS values increase overall ($K = 20.30, K = 30.32, K = 40.33, K = 50.35$), indicating more context clustering with increasing clusters. The plot of the the amount of clusters K vs PVE does not show a clear elbow shape like previous examples(Figure 14). By far the largest PVE is gained at $K = 2$. Still, the gain in PVE is significant when $K > 2$ and does not flatten as much as Figure 11 or Figure 12. The elbow point in Figure 14 is more ambiguous and open for interpretation. The spatial distribution of points provides a possible explanation. Clear sub clusters within clusters can be observed in Figure 13. Calculating the PVE values in other dimensions (actors, context, temporal) could provide a solution in this scenario.

Both the *spatial* and *context K-Means* clustering procedures provide solutions for clustering the #hagel collection. Which clustering result reflects events better? Both algorithms provide clusters with less variance than the original collection. In both case studies the kCS and kNN values improved. A hierarchy between dimensions might exist. In the current iteration of the methodology the hierarchy between dimensions is up to the user. Which hierarchy should be used when attempting to detect events in a similar fashion as the human brain? If the entire Tweet collection would be clustered using *spatial K-Means*, the clusters would indicate spatial hotspots of Tweets (two-dimensional) instead of events (multi-dimensional). Though the spatial dimension is an important dimension for events, it requires other dimensions to form event clusters. The *context* dimension encompasses many dimensions (as explored in subsection 3.1) making *context* clusters multi-dimensional by nature. To increase the certainty in event detection a combination of dimensions could be a strong indicator for events. This does require alteration of the *K-Means* algorithm as it is currently unable to calculate clusters in multiple dimensions.

The current clustering and classification procedures show a strong tendency to cluster areas which have a high Tweet density (see Figure 13, $SPK5C1$). Both *K-Means* and kNN assume a homogeneous area in terms of Tweet likelihood, a Tweet in a low density rural area has an equal chance of occurring as a Tweet in a high density urban area. Similar to the normalisation done by $tfidf$ for the textual dimension, the spatial distance needs to be normalized.

The context clusters as shown in Table 4 showed that CTK2C1 is a cluster with identical coordinates on every Tweet. This cluster has a strong spatial component ($kNN = 0$), but is not found by spatial clustering with low K values. The reason behind this is a limitation of the *K-Means* clustering algorithm as shown in Figure 15. However this problem can likely be mitigated by the inclusion of the temporal and actor dimension.

The *kNN* and *kCS* values are relative values that only have a real meaning in context. A *kNN* value of 2 does not indicate a strong or weak clustering as these terms are relative. It is therefore difficult to give a "human" interpretation of these values. The value of these measures lies in classification, for comparing and linking clusters and to support cluster decisions (ideal K value) in a similar way as the PVE.

4.5 Classify in multiple dimensions

In appendix D a classification of cluster SPK2C1 is shown, which has a spatial *centroid* calculated using the methodology as described in subsection 3.6. A spatial *volume*, which is the extent. It is calculated by taking the maximum distance between points and a spatial measure for *clustering* the *kNN*. In the context dimension the *centroid* is calculated in the same way (see subsection 3.6). The values are the mean of the *tfidf* vectors within the

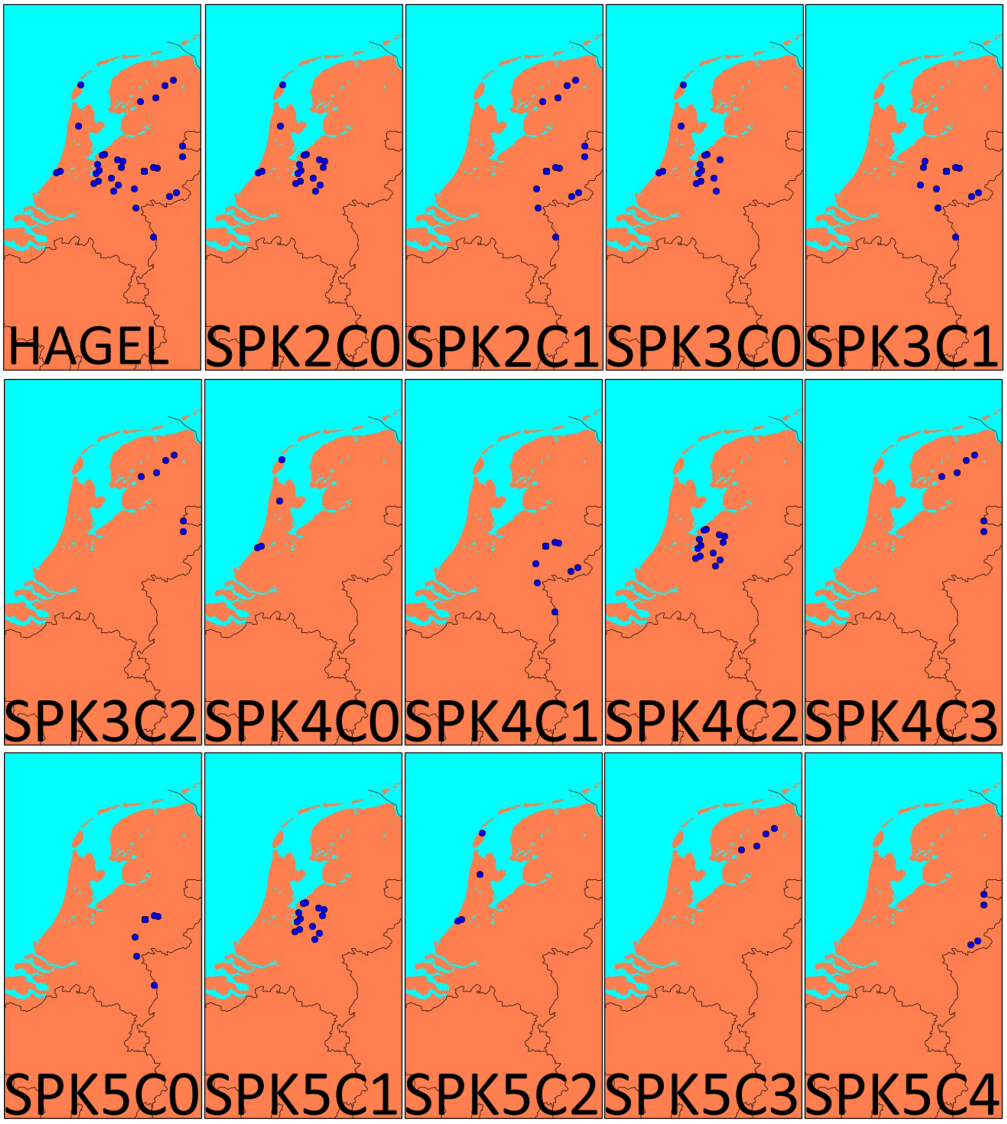

Figure 13: The data of Table 5 plotted on the map of the Netherlands showing the spatial distribution. The image labelled "HAGEL" is the initial collection, the other images are labelled with "SPKXCY", where X indicates the amount of clusters, and Y the cluster as a result of *K-Means* optimization.

Table 5: *K-Means* spatial clustering results ($K^I = 20, k = N$). K is the amount of clusters, PVE is the percentage of variance explained, the cluster refers to the cluster as a result of *K-Means* clustering. A visual spatial representation of each cluster can be found in Figure 13. The *kNN(sd)* is the k nearest neighbour statistic and its standard deviation, *kCS(sd)* is the k-nearest cosine similarity statistic and its standard deviation, N is the amount of Tweets in a cluster.

K	PVE	Cluster	kNN	kNNsd	kCS	kCSsd	N
1	0	$HAGEL$	2.966	1.288	0.256	0.115	53
2	0.539	$SPK2C0$	1.443	0.502	0.185	0.067	18
		$SPK2C1$	1.631	1.108	0.363	0.166	35
3	0.701	$SPK3C0$	1.35	0.453	0.157	0.053	15
		$SPK3C1$	0.818	0.634	0.417	0.169	32
		$SPK3C2$	0.933	0.17	0.171	0.066	6
4	0.806	$SPK4C0$	0.904	0.2	0.13	0.039	5
		$SPK4C1$	0.644	0.609	0.453	0.184	29
		$SPK4C2$	0.638	0.076	0.203	0.071	13
		$SPK4C3$	0.933	0.17	0.171	0.066	6
5	0.862	$SPK5C0$	0.461	0.574	0.505	0.179	27
		$SPK5C1$	0.638	0.076	0.203	0.071	13
		$SPK5C2$	0.904	0.2	0.13	0.039	5
		$SPK5C3$	0.375	0.07	0.082	0.013	4
		$SPK5C4$	0.558	0.051	0.265	0.1	4

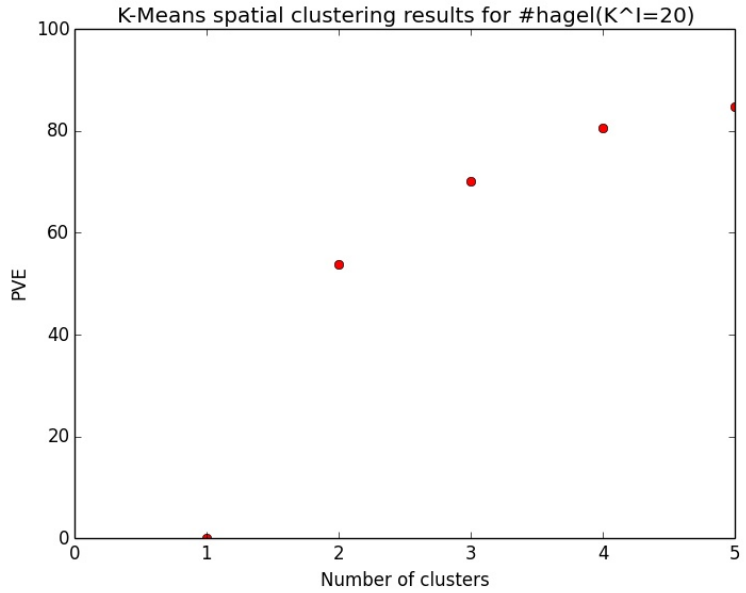

Figure 14: Elbow plot of the data in Table 5 the amount of clusters K on the X axis and the PVE value on the Y axis. The elbow point in this plot is difficult to identify.

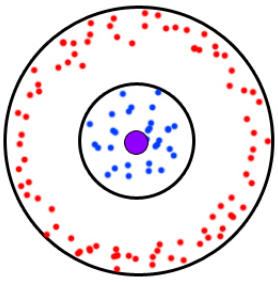

Figure 15: Given a hypothetical example of Tweets distributed in space. Human interpretation would classify this example as two clusters. The blue cluster which has a centroid in the middle (the purple dot) and a small distance from this centroid and a low *kNN* value. The red cluster which has its centroid in the middle as well (purple dot) but a large distance from this centroid and a large *kNN* value (if k=N). However, as the centroid is identical for both clusters *K-Means* clustering would not be able to differentiate between these clusters.

collection. The values behind each word give an indication on the importance of each word within the cluster. This collection is characterized especially by the terms: regen, onweer, apeldoorn, natte, sneeuw. The context *volume* is the amount of unique words (minus stop words) found in the collection and the *clustering* in the context dimension is the *kCS* value. The classification is a combination of identifiers which can be used to link events together. Mostly useful for use within computer algorithms but interpretable. The temporal and actor component are missing in the current implementation. Even though the use for the classification is currently limited, it can be seen as meta data of the clusters. Giving more information on the contents and homogeneity.

4.6 Methodology Discussion

The methodology is largely based on the underlying event definition as proposed in subsection 3.1. In this research evidence has been found in support of this definition. The multi-dimensional nature of events was shown by effectively clustering collections using these dimensions. Other research has shown similar results for the actor and temporal dimension. As such, the definition of an "event" as a multi-dimensional entity seems to work in the context of digital event detection. Analogous to event detection in the brain the proposed methodology attempts to create clusters based on similarity. This underlying principle seems to work for the case studies in this research, however, for true event detection a multi-dimensional approach needs to be taken.

The methodology in its current implementation add value to the data. The classified event clusters give insights in the data and allow interpretation of events as detected. This classification is crucial for any form of event detection, not in the least because of the arbitrary nature of events. The case study on classification in multiple dimensions shows that spatial clustering has its merits in event detection. Clusters in space can represent more optimized clusters in context. Unfortunately, clusters in space have difficulty with finding bots (due to a limitation of the K-Means algorithm Figure 15) which can have a large effect on the classification. Additionally, the importance of the spatial dimension for events varies between dimensions. As such, the underlying assumption of spatial dependence is not a rule to certain success for event detection (item (iii), 6).

The techniques and algorithms used can be easily applied to the temporal dimension, time already has an inherent distance metric. The actor dimension is more difficult to implement. Many aspects of this dimension can be measured: the user who Tweeted, the user mentioned using "@", a name in the context of a Tweet or event and/or the social stream as explored by Zhao et al. [61].

An overarching difficulty in event detection research is verification. Events are arbitrary classifications of happenings in space and time as such a "true" event does not exist. The methodology shows capability in performing its required features as stated in item (i). However the success at which these requirements are fulfilled is hard to measure. A human test panel could help in building a system that detects events as if it were human, unfortunately for this research such an experiment setup was not feasible. An alternative could be to check whether the events detected by the methodology also occur in the PR. However, before the effectiveness of clustering methodologies can be tested, more research is required on the correspondence between the DR (Twitter) and PR. This correspondence likely varies depending on the events. For example, events of broadcasting nature likely have a strong correspondence while discussions on a subject might only occur on Twitter and not in the PR. Another aspect is that the dimensions are not the same by definition, Tweets can be made by unrelated actors/hours later/miles away compared to the original event.

5 Conclusion

In this research a methodology and implementation has been created for multi-dimensional event detection on Twitter.

In related works, often an ad hoc definition is used for the term "event". In this research a multi-disciplinary definition is proposed, in which the event is defined as a multi-dimensional entity with four dimensions, each of these dimensions has three properties as shown in Figure 5. The event detection and classification as proposed in this methodology is based on this definition. This research made an implementation for the identification and classification of clusters in the spatial and textual dimension. Uncovering uncharted territory as this combination has barely been explored for event detection on Twitter (see Table 2).

Many approaches have been taken for Event Detection on Twitter. Often an underlying assumption of linearity and homogeneity of collections persists. In this research Twitter has been classified as a CAS using the methodology of Grus et al. [22], requiring a non-linear analysis approach. The complex nature suggests that Twitter does not behave in a linear fashion and assumptions of homogeneity is likely to produce errors.

BDA has been suggested as a complex data analysis to describe the events that occur on Twitter. The methodology is built up in the form of a knowledge pyramid, with stepwise value adding to the data (Figure 3). The basic components of the methodology consist of clustering and classifying. The process of clustering and classification is polymorph. The polymorph nature of the methodology is clear in both application and implementation. The methodology can be applied to detect any type of event, unlike event detection methodologies in related works which focus on events with specific dimension characteristics. The implementation is polymorph as the components are independent of dimension. The only required alteration is the distance metric. In this research it is shown that the methodology works in both the *spatial* and *context* dimension.

The implementation in its current form includes: procurement and preprocessing (Twitter REST API) distance measures (CS,Haversine), clustering (*K-Means*) and classification (kCS, kNN). These tools allow value adding from Big Data to information.

The results support the hypothesis of heterogeneity in the data. Within hashtag collections Tweets are found that belong to another collection, collections were clustered further into sub collections and a Twitter bot was identified using the clustering methodology. Clustering in both the spatial and context dimension led to intuitive results. However, the cluster detection implementation in this research is one-dimensional by nature. To identify events the multi-dimensional approach needs to be completed with the temporal and actor dimension. This could be done by considering PVE values in all dimensions to optimize clusters or by performing the *K-Means* algorithm on four dimensions at once. The clustering measures *kNN* and *kCS* can also serve as support in identifying clusters in which the Tweets show high dimension similarity. The elbow methodology for identifying the optimal (PVE vs K) amount of clusters (K) in a collection gave mixed results. For the context clusters a clear elbow point could be observed, the spatial clusters did not immediately lead to an unambiguous answer. Another downside of the elbow methodology is that it is supervised, requiring human interpretation. An automated alternative to the elbow method could be the gap statistic as proposed by Tibshirani et al. [51].

There is ambiguity in detected events using this methodology. The current methodology provided different clusters for the same base collection #hagel. Even when the clustering is done using both dimensions at the same time the algorithm needs to decide whether the spatial or context dimension is more important for the clustering procedure. A certain hierarchy in the importance of dimensions for event detection seems inevitable. However, this does make the classification biased. Perhaps this is inherent to "events" as Sakaki et al. [46]

("Events are arbitrary classifications of the space/time region") and Boyd and Crawford [10] ("Big Data analysis does not produce an unbiased objective information source") stated. When defining the mining parameters choices need to be made which by definition give a biased view on the data. These events are a result of the assumptions made when defining the methodology (item (iii)). No claim can be made that clusters found using this methodology are the "true" events. However, this methodology does offer an objective and repeatable approach to detection and classification of events. Parameter choices are up to the discretion of the user. Which is important as it makes the methodology polymorph in its applications.

The classification of clusters is an important step when moving from Small Data to Information. The classification is done using the *centroid*, *volume* and *clustering* in each dimension. This has been implemented for the spatial and context dimension. The classification gives the collections as a result from *K-Means* clustering an objective classification (see Appendix D). If fully implemented in all dimensions the classification can serve as an identifier for computer algorithms to calculate similarity between clusters, linking events together in a similar way as the works from Alvanaki et al. [3]. For example, the *volume* (amount of Tweets) over *time*, which in related works [35;14;54] is often referred to as bursts, is a an identifier for related collections. Using the classification, event clusters with similar *centroids* in the *spatial* and *temporal* dimension can be identified as related events.

To make an event detection methodology work in a multidimensional Big Data context, a framework has been designed in which every input and output is a query that described a data collection. The advantage of this approach is twofold: 1. Every component can communicate with other components. This is especially important when methodologies that analyse different dimensions need to communicate. 2. Saving queries instead of data object saves virtual memory. Which becomes increasingly important when the volume of data increases. In this research a first step is taken in optimizing the algorithms using queries. This can be further improved upon by using queries that incorporate the centroids of the data, only considering Tweets that are in the neighbourhood, approximately the same period or use words that are already present in the collection. Such a query structure could greatly reduce computing time.

The Nearest Neighbour methodology, used as a distance metric in *K-Means* and *kNN* does not work optimal in the context of Tweets. The underlying assumption of Spatial Dependence [38] (item (iii), point 3) only works when Tweets have a uniform likeliness of occurring in each location of the study area. A possible solution to circumvent this problem is by creating a statistical model of occurrence, similar to the *tfidf* methodology for context normalization.

Digital event detection research as explored in related works can easily implement elements of the proposed methodology to improve the value of their data and with that the value of their conclusions. The methodology can be used to extract events out of any dataset that has a context, spatial, temporal and/or actor component. For social studies, behaviour of Twitter users can be monitored and classified in a flexible manner, as this methodology allows event detection on any combination of dimensions. Similarities between dimensional patterns (i.e.: space/time, context/space events) can be quantified using the classification as proposed in the methodology, which allows patterns recognition of recurring events in time and space. The methodology has applications in disaster management by more accurately identifying relevant Tweets, using both the multi dimensional clustering and the classification. Anomaly detection can be improved using this methodology, as the multi dimensional nature of events more accurately indicates whether an event is an anomaly or not.

This research proposed a generic methodology for event detection, based on the multidimensional event definition and CAS/Knowledge Discovery theory. The main advantages of this methodology over methodologies in related works are:

(vii)
1. Identification of clusters in multiple dimensions which enables more flexibility and precision on which (type of) events are detected.
2. Objective multi-dimensional classification which is important for relating events and giving an interpretable insight into the data.
3. Polymorphism - the methodology can be applied on any type of event and is independent of dimension.

6 Bibliography

[1] F. Abel, C. Hauff, G. Houben, R. Stronkman, and K. Tao. Semantics+ filtering+ search= twitcident. exploring information in social web streams. In *Proceedings of the 23rd ACM conference on Hypertext and social media*, pages 285–294. ACM, 2012. ISBN 1450313353.

[2] Alexa.com. Alexa site info: Twitter. `http://www.alexa.com/siteinfo/twitter.com`, 2013. [Online; accessed 28-June-2013].

[3] F. Alvanaki, S. Michel, K. Ramamritham, and G. Weikum. See what's enblogue: real-time emergent topic identification in social media. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 336–347. ACM, 2012. ISBN 978-1-4503-0790-1. doi: 10.1145/2247596.2247636. URL `http://dl.acm.org/citation.cfm?id=2247596.2247636`.

[4] T.K. Anderson. Kernel density estimation and k-means clustering to profile road accident hotspots. *Accident Analysis & Prevention*, 41(3):359 – 364, 2009. ISSN 0001-4575. doi: http://dx.doi.org/10.1016/j.aap.2008.12.014. URL `http://www.sciencedirect.com/science/article/pii/S0001457508002340`.

[5] B Barnard. Wikipedia: Determining the number of clusters in a data set. `http://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set`, 2005. [Online; accessed 21-February-2014].

[6] E. Begoli and J. Horey. Design principles for effective knowledge discovery from big data. In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, pages 215–218. IEEE, 2012.

[7] G. Bellinger, D. Castro, and A. Mills. Data, information, knowledge, and wisdom. `http://www.systems-thinking.org/dikw/dikw.htm`, 2004. [Online; accessed 10-January-2014].

[8] S Bird, E Klein, and E Loper. *Natural Language Processing with Python*, pages 10–14. O'Reilly Media, 2nd edition, 2010. ISBN 978-0-596-51649-9. URL `http://nltk.org/book/`.

[9] J. Bollen, H. Mao, and X. Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1):1–8, 2011. ISSN 18777503 (ISSN). URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-79953102821&partnerID=40&md5=ea7c5bb18e5075c65d8c117f5c722300`.

[10] D. Boyd and K. Crawford. Six provocations for big data. In *A Decade in Internet Time: Symposium on the Dynamics of the Internet and Society, September 2011*. Microsoft Research, 2011. doi: 10.2139/ssrn.1926431. URL `http://papers.ssrn.com/abstract=1926431`.

[11] P.S. Bradley and U.M. Fayyad. Refining initial points for k-means clustering. In *ICML '98 Proceedings of the Fifteenth International Conference on Machine Learning*, volume 98, pages 91–99. Citeseer, 1998.

[12] W.F. Buckley. *Society–a Complex Adaptive System: Essays in Social Theory*, volume 9. Taylor & Francis, 1998.

[13] P.A. Burrough, P.F.M. van Gaans, and R.A. MacMillan. High-resolution landform classification using fuzzy k-means. *Fuzzy Sets and Systems*, 113(1):37 – 52, 2000. ISSN 0165-0114. doi: http://dx.doi.org/10.1016/S0165-0114(99)00011-1. URL http://www.sciencedirect.com/science/article/pii/S0165011499000111.

[14] M. Cataldi, L. Di Caro, and C. Schifanella. Emerging topic detection on twitter based on temporal and social terms evaluation. In *Proceedings of the Tenth International Workshop on Multimedia Data Mining*, page 4. ACM, 2010. ISBN 978-1-4503-0220-3. doi: 10.1145/1814245.1814249. URL http://dl.acm.org/citation.cfm?id=1814245.1814249.

[15] T. K. Das and P. Mohan Kumar. Big data analytics: A framework for unstructured data analysis. *International Journal of Engineering and Technology*, 5 (1):153–156, 2013. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84875345417&partnerID=40&md5=773695df98b4337f2ede7c540c8ce3cd.

[16] A. Dong, R. Zhang, P. Kolari, J. Bai, F. Diaz, Y. Chang, Z. Zheng, and H. Zha. Time is of the essence: improving recency ranking using twitter data. In *Proceedings of the 19th international conference on World wide web*, pages 331–340. ACM, 2010.

[17] G.H. Eoyang. *Coping with chaos: Seven simple tools*, page 196. Lagumo Corporation Cheyenne, WY, 1st edition, 1997.

[18] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge discovery and data mining: Towards a unifying framework. In *KDD*, volume 96, pages 82–88, 1996.

[19] D. Fisher, R. DeLine, M. Czerwinski, and S. Drucker. Interactions with big data analytics. *interactions*, 19(3):50–59, 2012. ISSN 1072-5520. doi: 10.1145/2168931.2168943. URL http://dl.acm.org/citation.cfm?id=2168931.2168943.

[20] Google. The google geocoding api. https://developers.google.com/maps/documentation/geocoding/, 2013. [Online; accessed 24-July-2013].

[21] K. Gordon. What is big data? *ITNOW*, 55(3):12–13, 2013.

[22] L. Grus, J. Crompvoets, and A.K. Bregt. Spatial data infrastructures as complex adaptive systems. *International Journal of Geographical Information Science*, 24(3):439–463, 2010.

[23] P.N. Howard, A. Duffy, D. Freelon, M. Hussain, W. Mari, and M. Mazaid. Opening closed regimes: what was the role of social media during the arab spring? http://pitpi.org/index.php/2011/09/11/, 2011. [Online; accessed 02-November-2013].

[24] Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651 – 666, 2010. ISSN 0167-8655. doi: http://dx.doi.org/10.1016/j.patrec.2009.09.011. URL http://www.sciencedirect.com/science/article/pii/S0167865509002323.

[25] A. Java, X. Song, T. Finin, and B. Tseng. Why we twitter: An analysis of a microblogging community, 12 August 2007 through 15 August 2007 2009. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-67650498181&partnerID=40&md5=772cb15017b591f08c871d89f1b759ef.

[26] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):881–892, Jul 2002. ISSN 0162-8828. doi: 10.1109/TPAMI.2002.1017616.

[27] R. Kelly. Twitter study reveals interesting results about usage - 40pointless babble. `http://www.pearanalytics.com/blog/2009/`, 2009. [Online; accessed 01-July-2013].

[28] W. Kelly. Unconventional uses of big data and predictive analytics | techrepublic. `http://www.techrepublic.com/blog/big-data-analytics/unconventional-uses-of-big-data-and-predictive-analytics/432`, 2013. [Online; accessed 04-July-2013].

[29] C.A. Kurby and J.M. Zacks. Segmentation in the perception and memory of events. *Trends in cognitive sciences*, 12(2):72–79, 2008.

[30] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600. ACM, 2010. ISBN 978-1-60558-799-8. doi: 10.1145/1772690.1772751. URL `http://dl.acm.org/citation.cfm?id=1772690.1772751`.

[31] K. Leetaru, S Wang, G. Cao, A. Padmanabhan, and E. Shook. Mapping the global twitter heartbeat: The geography of twitter. *First Monday*, 18(5), 2013. ISSN 13960466. URL `http://firstmonday.org/ojs/index.php/fm/article/view/4366`.

[32] R. Li, K. H. Lei, R. Khadiwala, and Chang. Tedas: a twitter based event detection and analysis system. *In 2012 IEEE 28th International Conference on Data Engineering (2012), pp. 1273-1276*, 2012. URL `http://www.citeulike.org/user/jasonandrewcook/article/11819705`.

[33] C. Lynch. Big data: How do your data grow? *Nature*, 455(7209):28–29, 2008. ISSN 0028-0836. doi: doi:10.1038/455028a. URL `http://www.nature.com/nature/journal/v455/n7209/full/455028a.html`. Nature Publishing Group.

[34] O. Maimon and M. Last. Knowledge discovery and data mining. *Kluwer Academic Publishers*, page 2, 2001.

[35] A. Marcus, M.S. Bernstein, O. Badar, D.R. Karger, S. Madden, and R.C. Miller. Twitinfo: aggregating and visualizing microblogs for event exploration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 227–236. ACM, 2011. ISBN 978-1-4503-0228-9. doi: 10.1145/1978942.1978975. URL `http://dl.acm.org/citation.cfm?id=1978942.1978975`.

[36] D. Metzler, S. Dumais, and C. Meek. Similarity measures for short segments of text. Technical report, Microsoft Research, Berlin, Heidelberg, 2007. URL `http://dl.acm.org/citation.cfm?id=1763653.1763660`.

[37] D. Metzler, C. Cai, and E. Hovy. Structured event retrieval over microblog archives. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 646–655. Association for Computational Linguistics, 2012. ISBN 978-1-937284-20-6. URL `http://dl.acm.org/citation.cfm?id=2382029.2382138`.

[38] Harvey J Miller. Geographic knowledge discovery. In S Shektar and H Xiong, editors, *Encyclopedia of GIS*, page 363. Springer US, 1st edition, 2008.

[39] H.J. Miller and J. Han. *Geographic data mining and knowledge discovery*, pages 10–13. Taylor & Francis Group, LLC, 2009.

[40] D. Newtson. Attribution and the unit of perception of ongoing behavior. *Journal of Personality and Social Psychology*, 28(1):28, 1973.

[41] Weston Pace. Wikipedia: K-means clustering algorithm. `http://en.wikipedia.org/wiki/K-means_clustering`, 2012. [Online; accessed 23-February-2014. The illustration was prepared with the Java applet, E.M. Mirkes, K-means and K-medoids. University of Leicester, 2011.].

[42] R. Procter, F. Vis, and A. Voss. Reading the riots on twitter: Methodological innovation for the analysis of big data. *International Journal of Social Research Methodology*, 16(3):197–214, 2013. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-84876022800&partnerID=40&md5=70477e7c7c1dd4423f778b6635c741f4`.

[43] Y. Raimond and S. Abdallah. Event ontology. `http://motools.sourceforge.net/event/event.html`, 2007. [Online; accessed 11-June-2013].

[44] Magnus Rosell. *Introduction to InformaInfo Retrieval and Text Clustering*, pages 6–7. KTH CSC, 2009. URL `http://www.nada.kth.se/~rosell/undervisning/sprakt/irintro090824.pdf`.

[45] J. Rowley. The wisdom hierarchy: Representations of the dikw hierarchy. *Journal of Information Science*, 33(2):163–180, 2007. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-33947173628&partnerID=40&md5=fa1b1dfc5eda039c159178ce6745476d`. cited By (since 1996)100.

[46] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, pages 851–860. ACM, 2010. ISBN 978-1-60558-799-8. doi: 10.1145/1772690.1772777. URL `http://dl.acm.org/citation.cfm?id=1772690.1772777`.

[47] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988. ISSN 0306-4573.

[48] P. Shrestha, C. Jacquin, and B. Daille. Clustering short text and its evaluation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7182 LNCS(PART 2):169–180, 2012. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-84858304474&partnerID=40&md5=f5b5e52e6e6ed4d3e4556c1749295e28`. cited By (since 1996)0.

[49] StatisticBrain.com. Twitter statistics. `http://www.statisticbrain.com/twitter-statistics`, 2014. [Online; accessed 13-February-2014].

[50] Y Sun, H Deng, and J Han. Probabilistic models for text mining. In Charu C. Aggarwal and ChengXiang Zhai, editors, *Mining Text Data*, pages 259–295. Springer US, 2012. ISBN 978-1-4614-3222-7. doi: 10.1007/978-1-4614-3223-4_8. URL `http://dx.doi.org/10.1007/978-1-4614-3223-4_8`.

[51] R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[52] Twitter. Twitter api documentation. `https://dev.twitter.com/docs/platform-objects/places`, 2013. [Online; accessed 22-December-2013].

[53] G. Vossen, D.E. Long, J.X. Yu, M. Nagarajan, K. Gomadam, A.P. Sheth, A. Ranabahu, R. Mutharaju, and A. Jadhav. *Spatio-Temporal-Thematic Analysis of Citizen Sensor Data: Challenges and Experiences*, volume 5802 of *Lecture Notes in Computer Science*, pages 539–553. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04408-3. doi: 10.1007/978-3-642-04409-0_52. URL `http://dx.doi.org/10.1007/978-3-642-04409-0_52`.

[54] J. Weng and B. Lee. Event detection in twitter. In *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media*, 2011.

[55] Wikipedia. Cosine similarity — wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Cosine_similarity`, 2013. [Online; accessed 22-December-2013].

[56] Wikipedia. Haversine formula — wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Haversine_formula`, 2014. [Online; accessed 8-January-2014].

[57] Wikipedia. K-means clustering algorithm — wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/K-means_clustering`, 2014. [Online; accessed 23-February-2014.].

[58] J.M. Wood. Understanding and computing cohen's kappa: A tutorial. *WebPsychEmpiricist Web Journal*, 2007.

[59] X. Wu, V. Kumar, Q.J. Ross, J. Ghosh, Q. Yang, H. Motoda, G.J. McLachlan, A. Ng, B. Liu, P.S. Yu, Z.-H. Zhou, M. Steinbach, D.J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-37549018049&partnerID=40&md5=95cc8d0cbb322c5af5da7ec01e4a9f08`. cited By (since 1996)471.

[60] J.M. Zacks, N.K. Speer, K.M. Swallow, T.S. Braver, and J.R. Reynolds. Event perception: a mind-brain perspective. *Psychological bulletin*, 133(2):273, 2007.

[61] Q. Zhao, P. Mitra, and B. Chen. Temporal and information flow based event detection from social text streams, 2007. 1619886 1501-1506.

[62] Arkaitz Zubiaga, Damiano Spina, VÃctor Fresno, and Raquel MartÃnez. Classifying trending topics: a typology of conversation triggers on twitter. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 2461–2464. ACM, 2011. ISBN 978-1-4503-0717-8. doi: 10.1145/2063576.2063992. URL `http://dl.acm.org/citation.cfm?id=2063576.2063992`.

A Database Attributes

Table 6: Attributes as saved in the MongoDB, their original JSON object, the type and an explanation about the attribute.

Attribute	Source JSON object	Type	Notes
id	object.id	LongInt	A unique identifier.
screen_name	object.author.screen_name	String	The screen name of the user who sent the Tweet
text	object.text	String	Text stored without preprocessing.
textVector	object.text	Dictionary	Text stored as vector using procedure (iv)
capital	object.text	List	The words, extracted from the Tweet text, that start with a capital but are not at the start of the sentence and are not a stopword
url	object.text	List	The websites extracted from the Tweet text.
time time	object.created_at	Datetime	The time at which the Tweet was sent
coordinates	object.coordinates	List [lat,lon]	Not all Tweets have coordinates. The JSON coordinates are converted from degrees to decimal.
place	object.place.name	String	The JSON place.name is a user specified entry which contains a toponym
placecoordinates	object.place.name	List [lat,lon]	The toponym is converted using a geocoder.
lang	object.lang	String	The language in which a Tweet is sent, user specified.

B Textual clustering table

Table 7: *K-Means* clustering textual results on a sub collection of the hashtag hagel($K^I = 20, k = 5$). K is the amount of clusters, PVE is the percentage of variance explained, the cluster refers to the cluster as a result of *K-Means* optimization, *kNN(sd)* is the k-nearest neighbour statistic and its standard deviation, *kCS(sd)* is the k-nearest cosine similarity statistic and its standard deviation

K	PVE	Cluster	kNN	kNNsd	kCS	kCSsd
1	0	0	0	0	0.702	0.012
2	0.446	0	0	0	0.74	0.014
		1	0	0	0.796	0.017
3	0.526	0	0	0	0.791	0.01
		1	0	0	0.866	0
		2	0	0	0.74	0.014
4	0.573	0	0	0	0.866	0
		1	0	0	0.744	0.014
		2	0	0	0.791	0.01
		3	-	-	-	-
5	0.605	0	0	0	0.866	0
		1	-	-	-	-
		2	-	-	-	-
		3	0	0	0.743	0.017
		4	0	0	0.791	0.01

C K-Means Top 15 words

Text collection 3: Top 15 most used words in a collection as a result of K-Means clustering of collection C1.

Cluster: C1A

{ns: 79, storm: 20, trein: 20, amsterdam: 10, treinverkeer: 8, vertraging: 8, rijden: 8, treinen: 8, weer: 7, thuis: 6, gaat: 5, wind: 5, centraal: 5, fail: 5, uur: 5}

Cluster: C1B

{peopleschoice: 100, breakoutartist: 91, austin: 70, mahone: 69, rt: 43, austinmahone: 21, ariana: 20, grande: 19, need: 16, vote: 15, love: 12, us: 11, get: 10, voting: 10, waiting: 9{

Text collection 4: Top 15 most used words in a collection as a result of K-Means clustering of collection C2.

Cluster: C2A

{ hagel: 53, onweer: 43, storm: 38, regen: 30, windstoten: 24, natte: 23, zie: 22, nederland: 22, mm: 22, apeldoorn: 22, sneeuw: 22, http://tco/xzm1bmcgdp: 22, coderood: 12, achtig: 10, codegeel: 10 }

Cluster: C2B

{ fyra: 113, vanaf: 112, geld: 86, http://tco/wqfqowmogc: 86, zie: 86, terug: 86, amsterdam: 81, fail: 76, breda: 63, centraal: 61, minuten: 58, vertraagd: 58, vandaag: 54, rijdt: 54, rotterdam: 53 {

Text collection 5: Top 15 most used words in a collection as a result of K-Means clustering of collection C3.

Cluster: C3A

{ sinterklaas: 239, storm: 31, pakjesavond: 30, wind: 24, hoor: 19, sint: 18, waait: 18, fijne: 17, bomen: 16, weer: 12, lekker: 10, wel: 10, zwartepiet: 10, vanavond: 9, coderood: 8}

Cluster: C3B

{acakfilm: 76, poin: 54, benar: 38, jawaban: 38, c:usa: 26, yg: 22, ada: 22, tdk: 22, fo: 8, c:uk: 7, 21: 4, tata: 4, 11: 4, idduy: 4 [hte: 4}

Text collection 6: Top 15 most used words in a collection as a result of K-Means clustering of collection C4.

Cluster: C4A

{ onweer: 84, storm: 61, hagel: 50, regen: 39, windstoten: 26, natte: 24, nederland: 24, sneeuw: 23, zie: 22, mm: 22, apeldoorn: 22, http://tco/xzm1bmcgdp: 22, coderood: 14, achtig: 10 codegeel: 10}

Cluster: C4B

{storm: 493, wind: 42, weer: 36, sinterklaas: 35, waait: 29, sint: 28, wel: 26, ns: 22, coderood: 22, huis: 22, bomen: 19, thuis: 18, hoor: 18, gaat: 17, pakjesavond: 17}

Text collection 7: Top 15 most used words in a collection as a result of K-Means

clustering of collection C5.

Cluster: C5A

{ pakjesavond: 138, sinterklaas: 33, fijne: 26, storm: 19, iedereen: 10, sint: 8, wij: 8, allemaal: 8, weer: 7, wind: 6, avond: 6, gezellig: 5, rt: 5, eten: 5, jaar: 5 }

Cluster: C5B

{sinterklaas: 213, storm: 26, wind: 23, hoor: 18, waait: 17, sint: 16, bomen: 15, weer: 12, lekker: 10, wel: 10, vanavond: 9, fijne: 8, zwartepiet: 8, coderood: 7, klaar: 7}

D Cluster Classification

Classification of cluster SPK2C1

Count:35
Spatial dimension
Centroid:[52.28, 6.10]
Extent:190km
Amount of clustering:1.631

Context dimension
Centroid: { *zware*: 0.2036, *zeg*: 0.1588, *achtig*: 2.4116, *koufront*: 0.2701, 4, 4*c*: 0.5401, *coderood*: 2.061, *eerste*: 0.1658, *hierzo*: 0.2637, *heerenveen*: 0.2287, *stormpiek*: 0.3097, 5, 1*c*: 0.2899, *midden*: 0.2271, 3, 0*c*: 0.3097, 5, 1: 0.7911, 5, 7: 0.2541, 11, 4: 0.2899, 3, 3: 0.2899, *piek*: 0.2701, 4, 3*c*: 0.2783, *dak*: 0.1818, 3, 8*c*: 0.3097, ??: 0.2585, *tekeer*: 0.2343, 6, 3*c*: 0.3097, 5, 7*c*: 0.3097, 2, 5*c*: 0.3097, *zoohee*: 0.3097, 12, 0: 0.3097, *laat*: 0.1524, *apeldoorn*: 4.4639, *kans*: 0.2116, 2, 8*c*: 0.3097, *wow*: 0.177, 4, 5*c*: 0.3097, 8, 4: 0.2585, *onweer*: 4.2622, *lekker*: 0.3419, *omroepgld*: 0.2899, *sint*: 0.1351, *uur*: 0.1389, *ineenkeer*: 0.2899, *super*: 0.1582, 3, 9: 0.3097, *volgt*: 0.2227, *http* : *//tco/g*0*dfh*5*usle*: 0.3097, *opeens*: 0.1991, 6, 1*c*: 0.3097, *hangt*: 0.2256, *presikhaaf*: 0.3097, 11, 1: 0.2783, *zitten*: 0.1667, *noodweer*: 0.2065, 13, 2: 0.3097, *nijmegen*: 0.2125, 13, 8: 0.2899, 2, 7: 0.2701, 5, 4: 0.2701, 050: 0.2899, *natte*: 4.5265, *felle*: 0.2899, 4, 5: 0.2899, *gaat*: 0.1256, 5, 0*c*: 0.2899, *zwaar*: 0.1985, 2, 6*c*: 0.3097, *slechter*: 0.2637, 15 : 40: 0.2783, *zie*: 3.1239, *storm*: 3.0355, *arme*: 0.2081, *gaan*: 0.1292, *stortbui*: 0.3097, *regen*: 7.1042, *nederland*: 3.5396, 4, 7*c*: 0.2899, 3, 3*c*: 0.3097, *kloosterhaar*: 0.3097, *passeert*: 0.2899, 2, 9*c*: 0.3097, *arnhem*: 0.2135, *vandaag*: 0.1282, *openstaan*: 0.3097, *mm*: 3.7485, *weertje*: 0.1831, *kapot*: 0.1765, *mt*: 0.2637, *zeer*: 0.1863, *sneeuw*: 3.9854, *http* : *//tco/crzdh*3*ezl*9: 0.3097, *hagel*: 5.3845, *codegeel*: 2.4389, *inclusief*: 0.2701, *buien*: 0.2541, 3, 9*c*: 0.5797, 3, 5*c*: 0.3097, *rond*: 0.1871, 9, 9: 0.2899, 10, 8: 0.3097, *leek*: 0.2412, *klap*: 0.2541, *windstoten*: 4.8373, *boven*: 0.1838, *fiets*: 0.1645, *weer*: 0.0972, 5, 5*c*: 0.2783, *raam*: 0.2107, *later*: 0.19, 5, 3*c*: 0.2783, 14, 4: 1.3185, *http* : *//tco/xzm*1*bmcgdp*: 4.8699, }
Extent:111 words
Amount of clustering:0.363

E Twitter feed algorithm

```
1 import tweepy
2 import os, sys
```

```
3  from geopy import geocoders
4  import json
5  from Tweet import Tweet
6  from mongoInterface import *
7
8  class CustomStreamListener(tweepy.StreamListener,mongoInt):
9      def __init__(self,api):
10         self.api=api
11         self.pymongoconn("dbname")
12
13     def on_status(self, status):
14         if status:
15             tweet = Tweet(status)
16             if tweet.writetodb:
17                 self.writetodb(tweet)
18             else:
19                 raise Error
20
21     def on_error(self, status_code):
22         print >> sys.stderr, 'Encountered error with status code:',
               status_code
23         return False # Kill the stream
24
25     def writetodb(self,tweet):
26         try:
27             data ={}
28             data['screen_name'] = tweet.name
29             data['time'] = tweet.time
30             data['coordinates'] = tweet.coordinates
31             data['placecoordinates'] = tweet.placecoordinates
32             data['coordinatesD'] = tweet.coordinatesD
33             data['placecoordinatesD'] = tweet.placecoordinatesD
34             data['id'] = tweet.id
35             data['lang'] = tweet.lang
36             data['source'] = tweet.source
37             data['text'] = tweet.text
38             data['place'] = tweet.place
39             data['excl'] = tweet.excl
40             data['ques'] = tweet.ques
41             data['url'] = tweet.urls
42             data['hash'] = tweet.hash
43             data['at'] = tweet.at
44             data['capital'] = tweet.capitals
45             data['textVector'] = tweet.textVector
46             #insert into db
47             self.tc.insert(data)
48         except AttributeError:
49             print "couldn't find", sys.exc_info()
50         except:
51             print "Got the following error: ",sys.exc_info()[0]
52
53 def login():
54     CONSUMER_KEY = ""
55     CONSUMER_SECRET = ""
56     ACCESS_TOKEN = ""
57     ACCESS_TOKEN_SECRET = ""
58
59     auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
60     auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)
61     return auth
62
63 def tweetstream():
```

```
64      try:
65          auth = login()
66          api = tweepy.API(auth)
67          streaming_api = tweepy.streaming.Stream(auth, CustomStreamListener(api
                ))
68          streaming_api.filter(follow=None, locations=[3.22,50.75,7.22,53.33])
69      except KeyboardInterrupt:
70          exitFlag = 0
```

Code 5: Twitter Feed program used to interact with the REST API

```
1  from __future__ import division
2  import math
3  import re
4  import sys
5  from geopy import geocoders
6  import shelve
7  import unicodedata
8  from collections import Counter
9  from nltk.corpus import stopwords
10
11 class Tweet(object):
12     def __init__(self,tweetobject):
13         self.locations = shelve.open("geocodes")
14         self.g = geocoders.GoogleV3() ##GoogleV3
15         self.text = tweetobject.text
16         self.time = tweetobject.created_at
17         self.place = tweetobject.place.name
18         self.name = tweetobject.author.screen_name
19         self.excl = self.findmark("!")
20         self.ques = self.findmark("?")
21         self.lang = tweetobject.lang
22         self.id = tweetobject.id
23         self.source = tweetobject.source
24         self.findurl()
25         self.findhash()
26         self.findat()
27         self.findnames()
28         self.findcoordinates(tweetobject)
29         self.checkcoordinates()
30         self.coordinatesD = self.convertdegrees(self.coordinates)
31         self.placecoordinatesD = self.convertdegrees(self.placecoordinates)
32         self.textVector = self.texToVector()
33
34     def findcoordinates(self,tweetobject):
35         if tweetobject.coordinates:
36             a,b=tweetobject.coordinates['coordinates']
37             self.coordinates =b,a
38         else:
39             self.coordinates = None
40         self.findcoords()
41
42     def findhash(self):
43         hashes=()
44         pattern = re.compile(r"(#[a-zA-Z0-9]+)")
45         for a in re.findall(pattern,self.text):
46             hashes+= (a,)
47         self.hash = hashes
48
49     def findat(self):
50         at=()
51         pattern = re.compile(r"(@[a-zA-Z0-9]+)")
52         for a in re.findall(pattern,self.text):
```

```
53              at+= (a,)
54          self.at = at
55
56      def findurl(self):
57          urls=()
58          pattern2 = re.compile(r"(?:www.|http[s]?://)(?:[a-zA-Z]|[0-9]|[$-_@
                .&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+")
59          for a in re.findall(pattern2,self.text):
60              urls+=(a,)
61          self.urls = urls
62
63      def findmark(self,sign):
64          mark=self.text.count(sign)
65          return mark
66
67      def findnames(self):
68          names = ()
69          pattern2=re.compile(r"(?!^)(?<![\!\.\?])([\s|\(])([A-Z][a-zA-Z\-]+?)
                ([\s|\.|\?|\!|\|\,\)])") ## (?<!|.|s) ?
70          c=re.findall(pattern2,self.text)
71          count = 0
72          try:
73              while c:
74                  for x,y,z in c:
75                      if self.text.find(y) !=0 and y not in names:
76                          names+=(y,)
77                      b = y.lower()
78                      self.text = self.text.replace(y,b,1)
79                  c=re.findall(pattern2,self.text)
80                  count+=1
81                  if count > 20:
82                      raise ValueError
83          except ValueError:
84              print "Endless loop"
85          except:
86              print sys.exc_info()[0], "error with findnames"
87          self.capitals = names
88
89      def findcoords(self ):
90          ## check for the Netherlands
91          if self.place != None:
92              self.place = unicodedata.normalize('NFKD', self.place).encode('
                    ascii','ignore')
93              try:
94                  if self.locations.has_key(str(self.place)):
95                      self.placecoordinates=self.locations[str(self.place)]
96                  else:
97                      x,y = self.g.geocode(str(self.place)+", The Netherlands")
                            #was geocode
98                      self.locations[str(self.place)]=y
99                      self.placecoordinates=self.locations[str(self.place)]
100             except ValueError:
101                 self.placecoordinates = None
102             except:
103                 self.placecoordinates = None
104             ## Check for Belgium
105             if self.placecoordinates == None:
106                 try:
107                     x,y = self.g.geocode(str(self.place)+", Belgium")
108                     self.locations[str(self.place)]=y
109                     self.placecoordinates=self.locations[str(self.place)]
110                 except ValueError:
```

```
111                     self.placecoordinates = None
112                 except:
113                     self.placecoordinates = None
114             if self.placecoordinates == None:
115                 self.locations[str(self.place)]=None
116 
117     def checkcoordinates(self):
118         self.writetodb = True
119         if not self.point_in_poly(self.placecoordinates):
120             if not self.point_in_poly(self.coordinates):
121                 self.writetodb = False
122         if self.time == None:
123             self.writetodb = False
124 
125 
126     def point_in_poly(self,coords):
127         inside = False
128         if coords != None:
129             y,x = coords
130             Netherlands = [(3.22,50.75),(3.22,53.33),(7.22,53.33),(7.22,50.75)
                    ]
131             n = len(Netherlands)
132             p1x,p1y = Netherlands[0]
133             for i in range(n+1):
134                 p2x,p2y = Netherlands[i % n]
135                 if y > min(p1y,p2y):
136                     if y <= max(p1y,p2y):
137                         if x <= max(p1x,p2x):
138                             if p1y != p2y:
139                                 xints = (y-p1y)*(p2x-p1x)/(p2y-p1y)+p1x
140                             if p1x == p2x or x <= xints:
141                                 inside = not inside
142                 p1x,p1y = p2x,p2y
143         return inside
144 
145     def convertdegrees(self,coords):
146         if coords != None:
147             lat,lon = coords
148             lat = lat * math.pi/180
149             lon = lon * math.pi/180
150             return lat,lon
151 
152     def textToVector(self):
153         text = self.stripRemove(self.text)
154         words = self.removestop(text)
155         return Counter(words)
156 
157     def removestop(self,words):
158         lijst = []
159         for item in words:
160             if item in stopwords.words('dutch') or item in stopwords.words('
                    english') or item == "i'm":
161                 pass
162             else:
163                 lijst.append(item)
164         return lijst
165 
166     def stripRemove(self,t):
167         lijst = []
168         tokens = t['text'].split()
169         for token in tokens:
170             if "#" not in token and "@"not in token:
```

```
171                 token = token.strip('\'".,?:-()|!')
172                 token = token.replace(".","")
173                 token = token.replace("$","")
174                 token = token.lower()
175                 if token != '' and len(token) > 1:
176                     lijst += [token]
177         for i in t['hash']:
178             i=i.strip("#")
179             lijst += [i]
180         for i in t['at']:
181             i=i.strip("@")
182             lijst += [i]
183         return lijst
```

Code 6: Tweet object used in the Twitter feed

F kNN algorithm

```
1  from __future__ import division
2  import math
3  import numpy as np
4  from mongoInterface import *
5
6  class kNN(mongoInt):
7      def calcANN(self):
8          self.DE = 0.5 / math.sqrt(self.n / self.A)
9          self.ANN = self.DO/self.DE
10         self.ANNsd = self.DOsd / self.DE
11         self.SE = 0.26136 / math.sqrt(self.n**2 / self.A)
12         self.ZANN = (self.DO - self.DE) / self.SE
13
14     def averageCol(self):
15         self.avgDistDict = {}
16         querydic  = {}
17         #compare each item against the rest of the collection.
18         for item in self.tc.find(self.query).batch_size(100):
19             querydic["id"] = item["id"]
20             self.avgDistDict[item["id"]] = [self.iterator(querydic)]
21         sumAvgDict = []
22         #loop over all k-average distances as given by the iterator
23         #move from dictionary to list because a dictionary can't be used for
               statistics
24         for item in self.avgDistDict:
25             sumAvgDict+=[self.avgDistDict[item]]
26         self.n = float(len(sumAvgDict))
27         arrAvgDict = np.array([sumAvgDict])
28         self.DO = np.mean(arrAvgDict)
29         self.DOmedian = np.median(arrAvgDict)
30         self.DOsd = np.std(arrAvgDict)
31         self.DOmin = np.min(arrAvgDict)
32         self.DOmax = np.max(arrAvgDict)
33
34
35     def makeCol(self):
36         ### query1 is the collection you want to add something of query2 to.
37         if self.query2 != None:
38             self.n=1
39             for item in self.tc.find(self.query).batch_size(100):
40                 self.queryIDs += [item["id"]]
41                 self.n +=1
42             self.distdic = {} ## dictionary with the kANN key:distance from
                   query1
```

```
43              querydic  = {}
44              for item in self.tc.find(self.query2).batch_size(100):
45                  if item["id"] not in self.queryIDs:
46                      querydic["id"] = item["id"]
47                      self.iterator(querydic)
48              sub =[]
49              ## calculate the kANN
50              for i in self.distdic:
51                  key,minD = self.distdic[i]
52                  self.DO = minD
53                  self.calcANN()
54                  if self.ANN < 1:
55                      sub += [self.tc.find_one({"id":key})["id"]]
56              self.colAdd = self.convertToCol(sub)
57          else:
58              print "give two queries for collection creation"
59
60      def convertToCol(self,lijst):
61          col = {}
62          col['id'] = {'$in':lijst}
63          return col
64
65  class skNN(kNN):
66      def __init__(self,query={},k=None,query2=None):
67          self.pymongoconn()
68          self.query = query
69          self.query2 = query2
70          self.k = k
71          self.A = 78078.0
72          self.queryIDs = []  ## capture the queryIDs so the iterator can't
                compare to it
73
74      def mainStat(self):
75          self.averageCol()
76          self.calcANN()
77
78      def iterator(self,querydic):
79          key=self.tc.find_one(querydic)
80          #check which coords to use
81          wc1 =self.whichcoordS(key)
82          distancelist = []
83          minimumd = 99999
84          for item in self.tc.find(self.query).batch_size(100):
85              if item["id"] != key["id"]:
86                  #check which coords to use
87                  wc2 =self.whichcoordS(item)
88                  d=self.haversine(key[wc1],item[wc2])
89                  distancelist += [d]
90          if self.k == None:
91              distancelist = np.array(distancelist)
92              minimumd = np.mean(distancelist)
93          else:
94              distancelist = sorted(distancelist)
95              minimumd = sum(distancelist[:self.k])/self.k
96          if self.query2 != None:
97              self.distdic[key["id"]] = key["id"],minimumd
98          else:
99              return minimumd
100
101     def haversine(self,pos1,pos2):
102         lat1,lon1 =pos1)
103         lat2,lon2 =pos2
```

```
104         a1 = (math.sin((lat2 - lat1)/2))**2
105         a2 = math.cos(lat1)* math.cos(lat2)*(math.sin((lon2 - lon1)/2))**2
106         a = math.sqrt(a1 + a2)
107         b = math.sqrt(1-a**2)
108         d = 2 * math.atan2(a,b)
109         distance2 = int(d * 6367)
110         return distance2
111
112     def convertdegrees(self,coords):
113         lat,lon = coords
114         lat = lat * math.pi/180
115         lon = lon * math.pi/180
116         return lat,lon
117
118     def whichcoordS(self,item):
119         if item["coordinates"] != None:
120             a= "coordinatesD"
121         elif item["placecoordinates"] != None:
122             a = "placecoordinatesD"
123         return a
```

Code 7: Algorithm for calculating the kNN

G kCS and tfidf algorithm

```
1  from __future__ import division
2  from textCollection import *
3  import re
4  from math import sqrt
5  from math import log1p
6  from collections import Counter
7  import numpy as np
8
9  class cosSim(textCol):
10     def __init__(self,query={},k=None,query2=None):
11         self.pymongoconn()
12         self.N = self.tc.count()
13         self.k = k
14         self.query = query
15         self.query2 = query2
16         self.queryIDs = []  ## capture the queryIDs so the iterator can't
               compare to it
17
18     def mainStat(self):
19         self.iteratorCos()
20         self.avgCosSim()
21
22     def avgCosSim(self):
23         avg = []
24         for item in self.cosDict:
25             if self.k == None:
26                 arr = np.array([self.cosDict[item]])
27                 avg += [np.mean(arr)]
28             else:
29                 lijst = sorted(self.cosDict[item],reverse=True)
30                 kdist = sum(lijst[:self.k])/self.k
31                 avg += [kdist]
32         average = np.array(avg)
33         self.ACS = np.mean(average)
34         self.ACSsd = np.std(average)
35
36     def iteratorCos(self):
```

```
37          self.vectorDict = {}
38          self.cosDict = {}
39          for item in self.tc.find(self.query).batch_size(100):
40              vec1 = item["tfidfVector"]
41              self.vectorDict[item["id"]] = vec1
42          for item in self.tc.find(self.query).batch_size(100):
43              vec1 = self.vectorDict[item["id"]]
44              self.cosDict.setdefault(item["id"],[])
45              for item2 in self.tc.find(self.query).batch_size(100):
46                  if item2["id"] != item["id"]:
47                      vec2 = self.vectorDict[item2["id"]]
48                      self.cosDict[item["id"]] += [self.getCos(vec1,vec2)]
49
50      def makeCol(self):
51          self.queryIDs = []
52          self.vectorDict = {}
53          self.cosDict = {}
54          for item in self.tc.find(self.query).batch_size(100):
55              self.queryIDs += [item["id"]]
56          self.distdic = {} ## dictionary with the kNN key:distance from query1
57          self.iteratorCosCol()
58          sub = []
59          for i in self.cosDict:
60              key,minCos = self.cosDict[i]
61              if minCos > 0.3:
62                  sub += [self.tc.find_one({"id":key})["id"]]
63          self.colAdd = self.convertToCol(sub)
64
65      def iteratorCosCol(self):
66          self.vectorDict = {}
67          for item in self.tc.find(self.query2).batch_size(100):
68              vec1 = item["tfidfVector"]
69              self.vectorDict[item["id"]] = vec1
70          for item2 in self.tc.find(self.query).batch_size(100):
71              vec1 = item2["tfidfVector"]
72              self.vectorDict[item2["id"]] = vec1
73          for item in self.tc.find(self.query2).batch_size(100):
74              if item["id"] not in self.queryIDs:
75                  vec1 = self.vectorDict[item["id"]]
76                  distancelist = []
77                  for item2 in self.tc.find(self.query).batch_size(100):
78                          vec2 = self.vectorDict[item2["id"]]
79                          distancelist +=[self.getCos(vec1,vec2)]
80                  distancelist = sorted(distancelist,reverse=True)
81                  minimumCos = sum(distancelist[:self.k])/len(distancelist[:self
                        .k])
82                  self.cosDict[item["id"]] = item["id"],minimumCos
83
84      def getCos(self,vec1, vec2):
85          intersection = set(vec1.keys()) & set(vec2.keys())
86          numerator = sum([vec1[x] * vec2[x] for x in intersection])
87          sum1 = sum([vec1[x]**2 for x in vec1.keys()])
88          sum2 = sum([vec2[x]**2 for x in vec2.keys()])
89          denominator = sqrt(sum1) * sqrt(sum2)
90          if not denominator:
91             return 0.0
92          else:
93             return float(numerator) / denominator
94
95      def convertToCol(self,lijst):
96          col = {}
97          col['id'] = {'$in':lijst}
```

```
98          return col
```

Code 8: Algorithm for calculating the kCS

```
1  import pymongo
2  import unicodedata
3  from collections import Counter
4  from nltk.corpus import stopwords
5  from math import log1p
6
7  class mongoInt(object):
8      def __init__(self,mdb="test3"):
9          self.mdb=mdb
10         self.pymongoconn()
11
12     def pymongoconn(self):
13         try:
14             conn=pymongo.MongoClient()
15             #print "Connected successfully to "+mdb+"!!!"
16             self.db = conn[self.mdb]
17             self.tc = self.db.tweets
18         except pymongo.errors.ConnectionFailure, e:
19             print "Could not connect to MongoDB: %s" % e
20
21 def createIDF(query):
22     a = mongoInt("tfidf")
23     b = mongoInt()
24     for i in b.tc.find(query).batch_size(100):
25         for j in i["textVector"]:
26             j = j.lower()
27             a.tc.update({"word":j},{"$inc":{"count":1}}, True, False)
28
29 def createTextVector(query):
30     b = mongoInt()
31     for i in b.tc.find(query).batch_size(100):
32         col =textToVector(i)
33         query = convertToCol([i["id"]])
34         b.tc.update(query,{"$set":{"textVector":col}})
35
36 def createTFIDFVector(query):
37     b = mongoInt()
38     N = b.tc.count()
39     c = mongoInt("tfidf")
40     for i in b.tc.find(query).batch_size(100):
41         col =tfidf(i["textVector"],c,N)
42         query = convertToCol([i["id"]])
43         b.tc.update(query,{"$set":{"tfidfVector":col}})
44
45 def tfidf(dic,dbconn,N):
46     for word in dic:
47         factor = dbconn.tc.find_one({"word":word})["count"]
48         dic[word] = dic[word] * log1p(N/factor)
49     return dic
50
51 def convertToCol(lijst):
52     col = {}
53     col['id'] = {'$in':lijst}
54     return col
55
56 def textToVector(text):
57     text = stripRemove(text)
58     words = removestop(text)
59     return Counter(words)
```

```
60
61 def removestop(words):
62     lijst = []
63     for item in words:
64         if item in stopwords.words('dutch') or item in stopwords.words('
               english') or item == "i'm":
65             pass
66         else:
67             lijst.append(item)
68     return lijst
69
70 def stripRemove(t):
71     lijst = []
72     tokens = t['text'].split()
73     for token in tokens:
74         if "#" not in token and "@"not in token:
75             token = token.strip('\'".,?:-()|!')
76             token = token.replace(".","")
77             token = token.replace("$","")
78             token = token.lower()
79             if token != '' and len(token) > 1:
80                 lijst += [token]
81     for i in t['hash']:
82         i=i.strip("#")
83         lijst += [i.lower()]
84     for i in t['at']:
85         i=i.strip("@")
86         lijst += [i.lower()]
87     return lijst
```

Code 9: Algorithm for calculating the tfidf values

H K-Means algorithm

```
1 from __future__ import division
2 import numpy as np
3 from mongoInterface import *
4
5 class kMeans(mongoInt):
6     def __init__(self,query={},k=2,cutoff = 0.01,text=True):
7         self.pymongoconn()
8         self.query = query
9         self.k = k
10        self.cutoff =  cutoff
11        self.text = text
12        self.main()
13
14    def main(self):
15        self.points = []
16        for item in self.tc.find(self.query).batch_size(100):
17            p=Point(item,self.text)
18            self.points += [p]
19        if self.text:
20            self.clusters = self.kMeansT(self.initialC())
21        else:
22            self.clusters = self.kMeansS(self.initialC())
23
24    #This method creates an initial set of clusters
25    def initialC(self):
26        cList = []
27        if self.text:
28            for p in self.points:
```

```
29                  cList += [p]
30              #create initial coords for the centroids
31              initial = random.sample(cList, self.k)
32              #check if these initual coords are not identical
33              u = getCosList(initial)
34              while u:
35                  initial = random.sample(cList, self.k)
36                  u = getCosList(initial)
37          else:
38              for p in self.points:
39                  cList += [p.coords]
40              unique_data = [list(x) for x in set(tuple(x) for x in cList)]
41              #take a random sample of points from the points equal to k.
42              initialCoords = random.sample(unique_data, self.k)
43              initial = []
44              for i in initialCoords:
45                  for p in self.points:
46                      if i == p.coords:
47                          initial += [p]
48                          break
49              #start with k clusters based on the initial value
50          clusters = [Cluster([p]) for p in initial]
51          return clusters
52
53      def kMeansS(self,clusters):
54          sent = True
55          while sent:
56              #take the centroid from the previous clusters and put them in a
                    dictionary
57              newPoints = dict([(c,[]) for c in clusters])
58              for p in self.points:
59                  # for each point (p) take the minimum distance to a cluster
60                  # the cluster that is closest is put in the cluster object
61                  cluster = min(clusters, key = lambda c:getDistance(p, c.
                        centroid))
62                  #add the point to the dictionary of cluster it is closests to
63                  newPoints[cluster].append(p)
64              biggest_shift = 0.0
65              #optimizing clusters in pseudo language
66              #for each cluster, if the cluster
67              for c in clusters:
68                  # if any new points are added to the dictionary that belongs
                        to the cluster
69                  if newPoints[c]:
70                      # add the points to the cluster and recaculate the
                            centroid
71                      # the shift is the distance from centroid to centroid
72                      shift = c.update(newPoints[c])
73                      biggest_shift = max(biggest_shift, shift)
74              #stop if no significant shifts occured
75              if biggest_shift < self.cutoff:
76                  sent = False
77          return clusters
78
79      def kMeansT(self,clusters):
80          sent = True
81          while sent:
82              #take the centroid from the previous clusters and put them in a
                    dictionary
83              newPoints = dict([(c,[]) for c in clusters])
84              for p in self.points:
85                  # for each point (p) take the minimum distance to a cluster
```

```
86                  # the cluster that is closest is put in the cluster object
87                  cluster = max(clusters, key = lambda c:getCos(p, c.centroid))
88                  #add the point to the dictionary of cluster it is closests to
89                  newPoints[cluster].append(p)
90              biggest_shift = 0.0
91              #optimizing clusters in pseudo language
92              #for each cluster, if the cluster
93              for c in clusters:
94                  # if any new points are added to the dictionary that belongs
                        to the cluster
95                  if newPoints[c]:
96                      # add the points to the cluster and recaculate the
                            centroid
97                      # the shift is the distance from centroid to centroid
98                      shift = c.update(newPoints[c])
99                      biggest_shift = max(biggest_shift, shift)
100             #stop if no significant shifts occured
101             if biggest_shift < self.cutoff:
102                 sent = False
103         return clusters
104
105 def getCosList(initialCoords):
106     for vec1 in initialCoords:
107         for vec2 in initialCoords:
108             if vec1.ref != vec2.ref:
109                 intersection = set(vec1.coords.keys()) & set(vec2.coords.keys
                        ())
110                 numerator = sum([vec1.coords[x] * vec2.coords[x] for x in
                        intersection])
111                 sum1 = sum([vec1.coords[x]**2 for x in vec1.coords.keys()])
112                 sum2 = sum([vec2.coords[x]**2 for x in vec2.coords.keys()])
113                 denominator = math.sqrt(sum1) * math.sqrt(sum2)
114                 if not denominator:
115                    return True
116     return False
117
118 def getCos(vec1, vec2):
119     intersection = set(vec1.coords.keys()) & set(vec2.coords.keys())
120     numerator = sum([vec1.coords[x] * vec2.coords[x] for x in intersection])
121     sum1 = sum([vec1.coords[x]**2 for x in vec1.coords.keys()])
122     sum2 = sum([vec2.coords[x]**2 for x in vec2.coords.keys()])
123     denominator = math.sqrt(sum1) * math.sqrt(sum2)
124     if not denominator:
125        return 0.0
126     else:
127        return float(numerator) / denominator
```

Code 10: MethodologyAlgorithm for calculating the K-Means

I Supporting algorithms for K-Means

```
1  from __future__ import division
2  import sys, math, random
3  from math import radians, cos, sin, asin, sqrt
4  from collections import Counter
5  from tkANN import *
6  from skANN import *
7  from cosSim import *
8
9
10 class Point:
11     def __init__(self, item,text):
```

```
12          self.text = text
13          if isinstance(item, dict):
14              if item.has_key("tfidfVector") :
15                  self.coordT = item["tfidfVector"]
16                  self.nT = len(self.coordT)
17              if item.has_key("coordinates"):
18                  if item["coordinates"] != None:
19                      self.coordS = item["coordinates"]
20                      self.nS = len(self.coordS)
21                  elif item.has_key("placecoordinates"):
22                      if item["placecoordinates"] != None:
23                          self.coordS = item["placecoordinates"]
24                          self.nS = len(self.coordS)
25              if item.has_key("id"):
26                  self.ref = item["id"]
27              else:
28                  self.ref = None
29          elif isinstance(item, Point):
30              self.coordT = item.coordT
31              self.coordS = item.coordS
32              self.ref = item.ref
33              self.nS = len(self.coordS)
34              self.nT = len(self.coordT)
35      def __repr__(self):
36          if self.text:
37              return str(self.coordT)
38          else:
39              return str(self.coordS)
40
41  class Cluster:
42      #input is a list of Points (class)
43      def __init__(self, points):
44          ## check if the points aren't empty
45          if len(points) == 0: raise Exception("ILLEGAL: empty cluster")
46          self.points = points
47          self.nS = points[0].nS
48          self.size = len(points)
49          self.text = points[0].text
50          if not self.text:
51              self.centroidS = self.calculateCentroidS()
52              self.centroidT = None
53          else:
54              self.centroidT = self.calculateCentroidT()
55              self.centroidS = None
56          self.col = {}
57          self.makeCol()
58
59      #if called, represented as:
60      def __repr__(self):
61          return str(self.points)
62
63      # update the centroid of a cluster to a new one and calculate the distance
            between centroids.
64      def update(self, points):
65          old_centroidS = self.centroidS
66          old_centroidT = self.centroidT
67          self.points = points
68          self.size = len(points)
69          self.centroidS = self.calculateCentroidS()
70          self.centroidT = self.calculateCentroidT()
71          if self.text:
72              distance = 1-getCos(old_centroidT.coordT, self.centroidT.coordT)
```

```
73          else:
74              distance = getDistance(old_centroidS, self.centroidS)
75
76          self.makeCol()
77          return distance
78
79      def makeCol(self):
80          self.col['id'] = {'$in':[]}
81          for p in self.points:
82              self.col['id']['$in'] += [p.ref]
83
84      def calculateCentroidS(self):
85          ## define a function to calculate the average x, and y coordintes
86          #self.points is the input for p, 0.0 is the starting input for x, the
                result for x is cumulative.
87          reduce_coord = lambda i:reduce(lambda x,p : x + p.coordS[i],self.
                points,0.0)
88          # devide the total of the "reduce coord"thus the total of lat or lon (
                i=0,i=1) by the amount of points.
89          centroid_coords = [reduce_coord(i)/len(self.points) for i in range(
                self.nS)]
90          cc = {}
91          cc["coordinates"] = centroid_coords
92          #convert the centroid to a point and return.
93          return Point(cc,False)
94
95
96      def calculateCentroidT(self):
97          o = {}
98          item = {}
99          length = len(self.points)
100         for i in self.points:
101             o = dict( (n, o.get(n, 0)+i.coordT.get(n, 0)) for n in set(o)|set(
                    i.coordT) )
102         for j in o:
103             o[j] = o[j] / length
104         item["tfidfVector"] = o
105         return Point(item, True)
106
107
108     def calculateCentroidT2(self):
109         distance = {}
110         pdict = {}
111         for i in self.points:
112             pdict[i.ref] = i
113             for j in self.points:
114                 distance.setdefault(i.ref,0)
115                 distance[i.ref] += (1 - getCos(i.coordT,j.coordT)) **2
116         centroid_id = min(distance, key=distance.get)
117         return Point(pdict[centroid_id],True)
118
119     def countwords(self):
120         self.vocabularyCount = Counter() #counts how many times a word occurs
                (actually the same as len for ID..)
121         for j in self.points:
122             for i in j.coordT:
123                 self.vocabularyCount[i] += 1#j.coords[i]
124
125     def statOfCol(self):
126         #self.time = tkANN(self.col,5)
127         #self.time.mainStat()
128         self.spat = skANN(self.col)
```

```
129         self.spat.mainStat()
130         self.txt = cosSim(self.col)
131         self.txt.mainStat()
132 
133     def sumOfSquares(self):
134         self.SS = 0
135         if self.text:
136             for i in self.points:
137                 self.SS += (1 - getCos(i.coordT,self.centroidT.coordT)) **2
138         else:
139             for i in self.points:
140                 self.SS += (getDistance(i,self.centroidS)) **2
141 
142     def produceCentroids(self):
143         self.extentS = 0
144         for i in self.points:
145             for j in self.points:
146                 distance = haversine(i.coordS,j.coordS)
147                 if distance > self.extentS:
148                     self.extentS = distance
149         print "\\textbf{Classification of cluster} \\\\"
150         print "\\textbf{Spatial dimension} \\\\"
151         print "\\textit{Centroid:}"+str(self.centroidS)+"\\\\"
152         print "\\textit{Extent:}"+str(self.extentS)+"km"+"\\\\"
153         print "\\textit{Amount of clustering:}"+str(self.spat.ANN)+"\\\\~\\\\"
154         print "\\textbf{Context dimension} \\\\"
155         print "\\textit{Centroid:} \{"
156         for i in self.centroidT.coordT:
157             print "$",i,"$:",
158             print str(round(self.centroidT.coordT[i],4))+", ",
159         print "\}\\\\"
160         print "\\textit{Extent:}"+str(len(self.centroidT.coordT))+" words"+"
            \\\\"
161         print "\\textit{Amount of clustering:}"+str(self.txt.ACS)+"\\\\"
162 
163 
164 
165 
166 ## input needs to be of point class
167 def getDistance(a, b):
168     if a.nS != b.nS: raise Exception("ILLEGAL: non comparable points")
169     ret = reduce(lambda x,y: x + pow((a.coordS[y]-b.coordS[y]), 2),range(a.nS)
        ,0.0)
170     return math.sqrt(ret)
171 
172 def haversine(pos1,pos2):
173     lat1,lon1 =pos1
174     lat2,lon2 =pos2
175     lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
176     a1 = (math.sin((lat2 - lat1)/2))**2
177     a2 = math.cos(lat1)* math.cos(lat2)*(math.sin((lon2 - lon1)/2))**2
178     a = math.sqrt(a1 + a2)
179     b = math.sqrt(1-a**2)
180     d = 2 * math.atan2(a,b)
181     distance2 = int(d * 6367)
182     return distance2
183 
184 def getCos(vec1, vec2):
185     intersection = set(vec1.keys()) & set(vec2.keys())
186     numerator = sum([vec1[x] * vec2[x] for x in intersection])
187     sum1 = sum([vec1[x]**2 for x in vec1.keys()])
188     sum2 = sum([vec2[x]**2 for x in vec2.keys()])
```

```
189     denominator = math.sqrt(sum1) * math.sqrt(sum2)
190     if not denominator:
191        return 0.0
192     else:
193        return float(numerator) / denominator
```

Code 11: Point and cluster objects as used by K-Means

```
1  def findIdealCentroid(self, k, txt, TSS, query, Ki):
2    #list with clisters
3      cL = {}
4      #amount of Cluters
5      aC = k
6      #Ki is the amount of times K-Means will be performed to find the ideal
           centroid.
7      for i in range(Ki):
8        #Initiate the WSS, start with 0, add to this object
9          WSS = 0
10         #perform kMeans for the amount of clusters aC, in eiter textual or
               spatial context txt is True or False
11         kM = kMeans(query,aC,text=txt)
12         for j in range(len(kM.clusters)):
13           #calculate the sumOfSquares for each cluster
14           kM.clusters[j].sumOfSquares()
15           #add the sum of squares to the WSS
16             WSS += kM.clusters[j].SS
17         #calculate the BSS by substracting the WSS from the WSS calculation
               when WSS =1, given as inputt.
18         BSS = TSS - WSS
19         #calculate PVE following the formula
20         PVE= BSS / self.TSS
21         #add the values to a list of clusters, which contains the kMeans
               object kM and values for PVE,WSS and BSS.
22         cL[i] = [kM,PVE,WSS,BSS]
23     #find the highest PVE value by looping through the cL.
24     maxPVE=0
25   for l in cL:
26       if cL[l][1] > maxPVE:
27         maxPVE = cL[l][1]
28         index = l
29     return {"kM":cL[index][0],"PVE":cL[index][1], "WSS":cL[index][2], "BSS":cL
           [index][3]}
```

Code 12: Algorithm to find the ideal centroid