FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

TweeProfiles2: real-time detection of spatio-temporal patterns in Twitter

João Henrique Alves Pereira

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Carlos Soares, PhD

Co-Supervisor: Tiago Cunha, MSc

June 24, 2014

TweeProfiles2: real-time detection of spatio-temporal patterns in Twitter

João Henrique Alves Pereira

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. Ana Paula Cunha da Rocha
External Examiner: Prof. Nuno Filipe Fonseca Vasconcelos Escudeiro
Supervisor: Prof. Carlos Manuel Milheiro de Oliveira Pinto Soares

June 24, 2014

Abstract

With the advent of social networking, a lot of user-specific, voluntarily provided data has been generated. A few years ago, people and companies started noticing the value that lied within those enormous amounts of information and started to think of ways to extract patterns from those and use them for gain.

TweeProfiles is a visualization tool that allows analysing tweets' data over 4 dimensions: spatial, temporal, social and content. It is, however limited in the sense that it is not prepared to deal with streaming data.

The goal of this work is to find, in real-time, patterns of information in data extracted from Twitter. To achieve this goal, a data mining process that is suitable for streaming data will be applied to the collected data. This process involves the data retrieval and pre-processing before it is passed to the clustering algorithm that will extract the desired patterns. The data to be harvested is multi-dimensional in nature, having namely a spatial dimension (the location of the tweet), a temporal dimension (the timestamp of the tweet) and a content dimension (the text of the tweet).

The resulting clusters can be presented to the end-user through a visualization tool that allows for an easier analysis and comprehension of the obtained results.

The tool was tested using a dataset from SocialBus with georeferenced tweets.

Resumo

Com o advento das redes sociais surgiu também muita informação, específica de cada utilizador e voluntariamente cedida. De há alguns anos para cá, as pessoas e as empresas começaram a perceber o valor que existe dentro dessas enormes quantidades de informação e começou a investigação em maneiras de encontrar padrões a partir desses dados que possam ser utilizados de forma proveitosa.

O TweeProfiles é uma ferramenta de visualização de clusters que permite a análise de tweets sob 4 dimensões: espacial, temporal, social e conteúdo. No entanto, é limitado no sentido em que não está preparado para lidar com streams de dados.

O objetivo deste trabalho é encontrar, em tempo real, padrões de informação em dados extraídos do Twitter. Para atingir este objectivo, foi desenvolvido um processo de *data mining* adequado para lidar com *streams* de dados para ser aplicado aos dados recolhidos. Este processo envolve a extração e o pré-processamento dos dados antes de serem passados para o algoritmo que irá extrair os padrões desejados. Os dados a serem colhidos são multi-dimensionais em natureza, ou seja, têm uma dimensão espacial (a localização do tweet), uma dimensão temporal (a hora do tweet) e uma dimensão de conteúdo (o texto do tweet).

Os clusters resultantes serão então apresentados ao utilizador final usando uma ferramenta de visualização que permite uma mais fácil análise e compreensão dos resultados obtidos.

Os resultados dos testes descritos foram obtidos utilizando um conjunto de dados com os tweets geo-referenciados, extraidos da plataforma SocialBus.

Acknowledgements

First and foremost, I would like to thank Prof. Carlos Soares for his supervision in this project, for his never-ending enthusiasm and for his valuable knowledge during the development of this dissertation.

Secondly, I would like to thank Tiago Cunha for all his patience and advice. His insights and points of view on the theme proved invaluable to the development of this work.

Next, I would like to thank my family for all the support shown during all the periods of my life, especially to my mother Conceição, my father Armando and my sister Gisela. For better or worse, I know I can always count on you.

Finally, I would like to thank all my friends, with a special remark to the ones from college, who made this journey a little bit easier. They say friends are the family you choose. I am very happy I chose all of you.

Porto, June 24th, 2014

João Henrique Alves Pereira

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Abbreviations

HMC	Hybrid micro cluster
MC	Micro cluster
PMC	Potential micro cluster
OMC	Outlier micro cluster

Chapter 1

Introduction

1.1 Context

Social networks play a big part in contemporary societies. Its influence is felt in several aspects of our lives, ranging from the way we do marketing, for instance, to the way we interact with our loved ones. From a business perspective, it has changed the way that companies are able to reach and capture their target audiencies. From a personal standpoint, it has changed the way people are able to get or stay in touch with their friends and family, even if they are not physically close. One business which has been significantly affected is journalism. Social networks can be used to change the way journalist are able to take the pulse of the trending themes or topics that are currently being talked about.

Twitter is one of the top social networks in existence, both in popularity (worldwide public awareness) and monthly active users (around 250 million [Twi14a]). The number of daily tweets also suffered a brutal evolution, growing from around 50 million in 2010 [Blo11] to around 500 million nowadays [Twi14a]. Adding that to a friendly and easy to use API, it makes Twitter a highly coveted target for developing R& D projects.

TweeProfiles [Cun13] is a Twitter analysis tool that enables the visualization of the results of a clustering methodology applied to a dataset of tweets. The data is processed over 4 dimensions of the data: spatial, temporal, social and content. It enables analysis giving different weights to each of the dimensions, producing different clustering models with the same dataset. It lacks, however, the ability to produce real-time visualizations of the evolution of the data stream, which with the growing volume of data can rapidly become an issue as it is unable to capture emerging trends in the patterns of the stream.

1.2 Motivation and Objectives

The main objective of this project is to develop a tool that allows real-time analysis of Twitter data. In order to achieve that, we will: 1) develop a method that allows the clustering of a stream of data; 2) develop a way to visualize the produced results; 3) test the algorithm and analise its

results. The selected algorithm must take the issues of volatility and multi-dimensionality of the content to be analyzed into account and successfully identify new emerging patterns and trends. Also, the visualization tool developed must allow for appropriate analysis of the results, namely geo-spatial analysis. The target audience for this project are Portuguese journalists, which stand to gain a helpful tool to aid them in assessing, in real-time, the topics that are currently trending in the twittosphere.

1.3 Project Description

The data mining process developed in this project consists of a clustering algorithm and a tool to visualize the output clusters. The tasks accomplished along the process are: data retrieval, data pre-processing, data clustering (involves distance calculation, combination and normalization) and results visualization.

The clustering algorithm used was a modified version of the DenStream framework, adapted to deal with text instances, as well as numeric ones. The DenStream framework is divided in two distinct steps: the online step, which summarizes the input stream into micro-cluster structures, and the offline step, which groups micro-clusters to produce the final clustering results. The most important modification is the distance function of the algorithm, which was altered to compute a composite distance between instances. The components of the modified distance function include the haversine distance (great-circle distance between geo-located points), the euclidean distance and the cosine similarity (dissimilarity measure between texts).

The test dataset was obtained from SocialBus, from which we extracted a set of about 50.000 geo-referenced tweets. This data contains information about the dimensions that are the target of our study (spatial, temporal and content) but has to undergo a pre-processing step before being passed along to the clustering algorithm.

The resulting clusters can be visualized as points in a world map, located according to their center; also, for each cluster, the user can see its most important statistics (such as the center, radius and weight of a cluster), as well as a word cloud contain the most frequently mentioned words in the tweets in the cluster.

1.4 Document Structure

This document has the following structure: Chapter 2 summarizes the state of the art of the scientific themes addressed by this project, namely: Twitter's functions and concepts; stream clustering algorithms; and inter-cluster distance measures. Chapter 3 explains the whole functioning and architecture of the developed tool. In chapter 4 the testing setup and the clustering results obtained in the tests are explained and analyzed. In chapter 5 a balance is made about the work done, mentioning the conclusions obtained and the work to be done.

Chapter 2

State of the art

In this chapter we will review the state of the art in the scientific domain of our project. In the sections 2.1 and 2.2 we will detail more techical aspect related to the clustering processes and algorithms, and the similarity (distance) functions, respectively. In section 2.3 we will give and overview of Twitter and TweeProfiles.

2.1 Clustering

Data mining is the process of discovering interesting patterns and knowledge from large amounts of data [HKP11]. However, as the authors further state, there is a broader definition for the term data mining, when it is "used to refer to the entire knowledge discovery process". The steps of the knowledge discovery process are [HKP11]:

1. Data cleaning (to remove noise and inconsistent data)
2. Data integration (where multiple data sources may be combined)
3. Data selection (where data relevant to the analysis task are retrieved from the database)
4. Data transformation (where data are transformed or consolidated into forms appropriate for mining by performing summary or aggregation operations, for instance)
5. Data mining (an essential process where intelligent methods are applied in order to extract data patterns)
6. Pattern evaluation (to identify the truly interesting patterns representing knowledge based on some interestingness measures)
7. Knowledge presentation (where visualization and knowledge representation techniques are used to present the mined knowledge to the user)

Clustering is a data mining task which falls into the category of unsupervised learning type of data mining methods meaning it is “a form of learning by observation, rather than learning by examples” [HKP11]. It is the process of partitioning a set of data objects into subsets such that objects in a cluster are similar to one another, yet dissimilar to objects in other clusters.

The level of similarity between evaluated objects depends on the distance function used, and different clustering methods use different distance functions, meaning that different methods and algorithms applied to the same dataset may generate different clusters. The measure of dissimilarity between objects is called distance, and there are several distance measures that can be used, which will be addressed in section 2.2.

There are four main types of clustering methods: partitioning, hierarchical, density-based and grid-based. K-means, BIRCH and DBSCAN (and its variations), among others, are some examples of clustering algorithms. We will explain DBSCAN in more detail as it used in the clustering process of our work (see chapter 3).

DBSCAN is a density-based clustering algorithm first proposed in 1996 [EKSX96]. It has two main parameters ε and *minPts* and is based on the concepts of ε-neighbourhood, density-reachability and core object. The ε-neighbourhood of a point p is considered to be the set of points which is not farther than ε from p. A point p_1 is considered density-reachable from another point p_n if there is a sequence of points $p_1 \dots p_n$ such that p_i is in the ε-neighbourhood of p_{i+1}; A core object is a point which has a ε-neighbourhood with at least *minPts* points.

DBSCAN forms a cluster by selecting a core object and adding its ε-neighbourhood to the cluster. Then, iteratively, visit all the points in the cluster; if any of them are also core objects, add their ε-neighbourhoods to the cluster, stopping when all the points in the cluster are visited.

2.1.1 Stream Clustering

As Gama [Gam10] states, the world of today is "a world in movement", where the data we seek to analyze and extract information from has new characteristics:

1. Data is made available through *unlimited streams* that continuously flow, eventually at high speed, over time;
2. The underlying *regularities may evolve over time* rather than be stationary;
3. The data can no longer be considered *independent and identically distributed*;
4. The data is now often *spatially situated as well as time situated*;

To address the volatility and quantity of the data that is the aim of this project, one must resort to using a special type of clustering algorithms, which are designed to treat streams of data.These are called stream clustering algorithms and differ from the regular clustering algorithms in that they must fulfill certain requirements. Barbara [Bar02] states that these requirements are:

1. Compactness of representation;
2. Fast, incremental processing of new data points;

3. Clear and fast identification of outliers;

The points mentioned above are all very pertinent and pose very important questions regarding the main issues with stream clustering. This means that:

- if the representation of the processed data is very thorough and precise, it will also require a great deal of resources (disk space and processing power) that with the arrival of new data may rapidly become impractical;
- if the processing of the new data points takes a long time, it will defeat the purpose of real-time analysis, as it will reach a point that new data will have arrived before the previous batch is processed, so, usually, stream clustering algorithms are required to process new batches of data in one pass;
- as data streams are usually very volatile in nature, we must have an effective way to distinguish between what is an outlier in the current clustering model and what is a new trend that is emerging in the stream, with the latter requiring a change in the clustering model that is currently being considered.

Figure 2.1 illustrates these requirements by comparing traditional and stream data processing characteristics.

	Traditional	**Stream**
Number of passes	Multiple	Single
Processing Time	Unlimited	Restricted
Memory Usage	Unlimited	Restricted
Type of Result	Accurate	Approximate
Distributed?	No	Yes

Figure 2.1: Differences between traditional and stream data processing [Gam10]

Now that we have mentioned the main differences between regular and stream clustering methods, we will dig deeper into some of the existing stream clustering algorithms.

2.1.1.1 STREAM

STREAM was first proposed by Guha [GMMO00]. It is a partitioning algorithm based on the popular k-means algorithm. In partitioning algorithms, the aim is to divide the input dataset into k partitions, each one representing a cluster of data points. The clusters are formed by calculating a set of k centroids, which are points representative of each cluster and then assigning the remaining data points to its closest centroid. STREAM works as follows:

1. The stream to be clustered is divided to chunks of m data points.

2. For each chunk, we pick k representatives, and assign each point of the chunk to its closest representative. The objective is to pick the k representatives in such a way that the sum of squared distances from each point to its assigned representative is minimal.

3. After each chunk is processed, the information about the k representatives of that chunk is stored, along with its weight, which is the number of points that was assigned to each representative. These are considered $level$-1 representatives.

4. When the number of stored representatives exceeds m (the size of a chunk), they are re-clustered, only this time their weights are factored in the clustering process. The resulting representatives are considered to be $level$-2 representatives. This process goes on, meaning when the number of $level$-n representatives exceeds m, they are clustered producing k $level$-$(n+1)$ new representatives.

5. Finally, when the entire stream is processed, one final pass is applied to the remaining representatives of all levels, clustering them together.

The major flaw in STREAM is its low sensitivity to the emerging trends in the processed stream. As we mentioned before, most data streams are very volatile and ever-changing in nature, which means this is a big drawback of this algorithm.

2.1.1.2 CluStream

CluStream was introduced by Aggarwal [AHWY03]. It is, like STREAM, a partitioning algorithm, but unlike it, it has two components: an online, micro-clustering component and an offline, macro-clustering one.

The micro-clustering component summarizes the input streaming data into micro-clusters, reducing the amount of data points that is to be used by the offline component. These micro-clusters are saved at moments in time that follow pyramidal time frame, a technique that provides a cost-effective way of maintain time granularity data about each micro-cluster.

> For a set of d-dimensional points $X_{i_1} \ldots X_{i_n}$ with time stamps $T_{i_1} \ldots T_{i_n}$ a micro-cluster is defined as the $(2 \cdot d + 3)$ tuple $(\overline{CF2^x}, \overline{CF1^x}, CF2^t, CF1^t, n)$ [AHWY03], where:
>
> - $\overline{CF2^x}$ is a d-sized vector containg the sum of squares of the data values for each dimension
> - $\overline{CF1^x}$ is a d-sized vector containg the sum of the data values for each dimension
> - $CF2^t$ is the sum of the squares of the time stamps
> - $CF1^t$ is the sum of the time stamps
> - n is the number of data points in the micro-cluster

These micro-clusters are generated in real time and stored at certain moments in time, as "snapshots" of the clustering model, so that later they can be used by the offline component to reconstruct or approximate the state of the model at any given point in time. The moments in time in which the snapshots are saved follow a pyramidal pattern, which provides an effective trade-off between the storage requirements and the ability to recall summary statistics from different time horizons. This is the technique known as pyramidal time frame which organizes snapshots into different orders according to the time granularity at which they are to be maintained. The order of a snapshot varies from 1 to $log(T)$, where T is the clock time elapsed since the beginning of the stream. The snapshots of the i-th order are stored every α^i time intervals (α is an integer bigger than 1) with only the last $\alpha + 1$ snapshots of each order being kept in memory.

Now that we established the two most important concepts introduced by CluStream, let us understand how micro-clusters are generated and maintained. Typically, when a new data point arrives, it is aggregated to one of the existing micro-clusters. However, when it falls outside the maximum boundary of the nearest micro-cluster (whether it is an outlier or the start of a new cluster) the new data point must be put in its own, new micro-cluster. In order to do that, and to keep the number of micro-clusters constant the set of previous of micro-clusters must be reduced by one. This can be achieved in one of two ways: either a cluster is deleted, or two are merged. That is decided by calculating the "relevance stamp" of each micro-cluster, which is a measure of time recency. If any of the values fall below a user pre-defined threshold, that cluster can be safely removed to give way to the new one. Otherwise, the two nearest clusters are merged.

The exposed features of CluStream make it a good candidate for clustering streams as they prove it encompasses mechanisms for factoring in the evolution characteristics of streams. However, there are still some issues with this algorithm (and most partitioning algorithms), namely one of its parameters being the number of (micro) clusters to be formed. When dealing with uncertain data streams, this parameter is one that we cannot know *apriori* and sometimes an "incorrect" value may bias the clustering models achieved.

2.1.1.3 DenStream

Described in [CEQZ06], DenStream is a density-based stream clustering algorithm. This means that, unlike partitioning methods, which are almost always based on distances, and therefore can only find spherical clusters, in density-based methods the "general idea is to continue growing a given cluster as long as the density (number of objects or data points) in the neighborhood exceeds some threshold" [HKP11], allowing for the formation of arbitrarily shaped clusters. Also, density-based methods do not require the apriori definition of the number of clusters to be found.

Density areas in regular clustering algorithms are represented by all points belonging to the cluster; as this would be impractical to do when dealing with a stream, DenStream applies the concept of core-micro-clusters.

> A core-micro-cluster, at time t is defined as $CMC(w, c, r)$ for a group of close points $X_{i_1} \dots X_{i_n}$ with time stamps $T_{i_1} \dots T_{i_n}$:

- $w = \sum_{j=1}^{n} f(t - T_{i_j}), w > \mu$ is the weight;
- $c = \frac{\sum_{j=1}^{n} f(t-T_{i_j}) X_{i_j}}{w}$ is the center;
- $r = \frac{\sum_{j=1}^{n} f(t-T_{i_j}) dist(X_{i_j}, c)}{w}, r \leq \varepsilon$ is the radius, where $dist(X_{i_j}, c)$ denotes the Euclidean distance between the point X_i and the center c [CEQZ06]

The temporal decay of a data point is given by the function

$$f(t) = 2^{-\lambda \cdot t}, \lambda > 0 \tag{2.1}$$

which means that the older the point is, the less weight it has.

However, when a new data points arrives, it is unlikely that the newly created micro-clusters obeys the weight constraint. So, the concepts of potential core-micro-cluster (p-micro-cluster) and outlier-micro-cluster (o-micro-cluster) are also introduced. These are defined as regular core-micro-cluster, with the only difference being the relaxation of the weight constraint, which is $w \geq \beta \cdot \mu$ for p-micro-clusters and $w < \beta \cdot \mu$ for o-micro-clusters, with $0 < \beta < 1$.

As a new data point arrives, it is merged with one of the existing p-micro-clusters. If it is not possible, because it violates the radius constraint, the next step is to try to merge it with one of the existing o-micro-clusters. If it is successfully merged, the weigth of that cluster is recalculated and depending on the satisfaction of the expresison $w \geq \beta \cdot \mu$, the o-micro-cluster is converted into a new p-micro-cluster. Otherwise, if the new data point cannot be merged with any of the existing micro-clusters, a new o-micro-cluster is created containing solely the new data point.

As time goes by, the weigths of the older micro-clusters decay. Every T_p time periods, those clusters whose weight does not reach the threshold ξ are deleted. This threshold is given by:

$$\xi(t_c, t_o) = \frac{2^{-\lambda \cdot (t_c - t_o + T_p)} - 1}{2^{-\lambda \cdot T_p} - 1} \tag{2.2}$$

where t_c is the current time and t_o is the time of creation of the micro-cluster.

In order to obtain the final clustering results, an offline component uses a variant of the DBSCAN algorithm to connect the stream density areas represented by the micro-clusters generated by the online component, producing the arbitraly shaped set of clusters.

2.1.1.4 Other algorithms

In [SFB+13], a survey of stream clustering algorithms is conducted, detailing 13 major stream clustering algorithms there are, including the ones detailed above. Several important aspects of stream clustering are referred and compared, namely the cluster (and/or micro-cluster) structures used by each algorithm, the temporal decay approaches, or the applicantions they were used in, among others. Figure 2.2 shows the list of the algorithms analyzed [SFB+13].

Algorithm	Data Structure	Window Models	Outlier Detection	Number of Parameters
(1) *BIRCH*	feature vector	landmark	density-based	5
(2) *CluStream*	feature vector	landmark	statistical-based	9
(3) *ClusTree*	feature vector	damped	—	3
(4) *D-Stream*	grid	damped	density-based	5
(5) *DenStream*	feature vector	damped	density-based	4
(6) *DGClus*	grid	landmark	—	5
(7) *ODAC*	correlation matrix	landmark	—	3
(8) *Scalable k-means*	feature vector	landmark	—	5
(9) *Single-pass k-means*	feature vector	landmark	—	2
(10) *Stream*	prototype array	landmark	—	3
(11) *Stream LSearch*	prototype array	landmark	—	2
(12) *StreamKM++*	coreset tree	landmark	—	3
(13) *SWClustering*	feature vector	landmark	—	5

Algorithm	Cluster Algorithm	Cluster Shape	Cluster Problem
(1) *BIRCH*	*k*-means	hyper-sphere	object
(2) *CluStream*	*k*-means	hyper-sphere	object
(3) *ClusTree*	*k*-means/*DBSCAN*	arbitrary	object
(4) *D-Stream*	*DBSCAN*	arbitrary	object
(5) *DenStream*	*DBSCAN*	arbitrary	object
(6) *DGClust*	*k*-means	hyper-sphere	attribute
(7) *ODAC*	hierarchical clustering	hyper-ellipsis	attribute
(8) *Scalable k-means*	*k*-means	hyper-sphere	object
(9) *Single-pass k-means*	*k*-means	hyper-sphere	object
(10) *Stream*	*k*-median	hyper-sphere	object
(11) *Stream LSearch*	*k*-median	hyper-sphere	object
(12) *StreamKM++*	*k*-means	hyper-sphere	object
(13) *SWClustering*	*k*-means	hyper-sphere	object

Figure 2.2: Characterization of existing stream clustering algorithms [SFB+13]

2.2 Distance Measures

As it was already mentioned above, in the context of clustering, there are various kinds of distance measures that can be used, depending on the data we are processing. These functions are used to calculate the similarity between objects.

2.2.1 Numerical Distance

As per Han [HKP11], the most popular distance measure is the Euclidean Distance (also known as straight line distance). If we consider two objects, x_1 and x_2, composed of n attributes each, the Euclidean distance between them is defined as:

$$dist(X_1, X_2) = \sqrt{(X_{1_1} - X_{1_2})^2 + \cdots + (X_{1_n} - X_{2_n})^2} \tag{2.3}$$

Also a very well known and often used distance, the Manhattan (or block) Distance is named in such a way since it gives us a distance in blocks between two points in a city (for example, 2

blocks down and 3 blocks over is a total distance of 5 blocks). Its formula is as follows:

$$dist(X_1, X_2) = |X_{1_1} - X_{1_2}| + \cdots + |X_{1_n} - X_{2_n}| \tag{2.4}$$

As for the Minkowski Distance, it is no more than a generalization of both the Euclidean and Manhattan Distances:

$$dist(X_1, X_2) = \sqrt[p]{|X_{1_1} - X_{1_2}|^p + \cdots + |X_{1_n} - X_{2_n}|^p}, p \geq 1 \tag{2.5}$$

It is also called the $L_p norm$ because of the notation of p in the formula. When $p = 1$ it represents the Manhattan Distance ($L_1 norm$) and when $p = 2$ represents the Euclidean Distance ($L_2 norm$). Lastly, we also have the Chebysev Distance or supremum distance (also known as $Lmax$ or $L_\infty norm$). It is a generalization of the Minkowski formula for $p = \infty$:

$$dist(X_1, X_2) = \lim_{p \to \infty} (\sum_{j=1}^{n} |X_{1_j} - X_{2_j}|^p)^{\frac{1}{p}} = max_j |X_{1_j} - X_{2_j}| \tag{2.6}$$

This formula gives us the maximum distance between the values of each attribute of the objects.

The Haversine Distance [Han76] is used to calculate the great-circle distance between two points, based on their geographical coordinates(latitude and longitude). It is an aproximate distance (it has a high accuracy, though) because it assumes the Earth is a perfect sphere, when in reality is slightly flattened in the poles. The haverine formula is:

$$distSp(X_1, X_2) = 2 \cdot R \cdot \sin^{-1}\left(\left[\sin^2\left(\frac{\theta_{X_1} - \theta_{X_2}}{2}\right) + \cos\theta_{X_1} \cdot \cos\theta_{X_2} \cdot \sin^2\left(\frac{\lambda_{X_1} - \lambda_{X_2}}{2}\right)\right]^{0.5}\right) \tag{2.7}$$

where R is Earth's radius,θ e point's latitude and λ the point's longitude.

2.2.2 Textual Distance

As mentioned by Han [HKP11], the most commonly used measure for comparing text documents is the cosine similarity. However, in order to do that, those documents must be represented in the form of a frequency vector, which is a data structure that associates a list of terms with a frequency measure of each term in relation to a text. This type of document representation is called *bag of words*. The absolute frequency of each term is usually used in the *bag-of-words* model; however *tf-idf* (term frequency-inverse document frequency) [MRS08] is also a commonly used term-frequency measure. If you consider a tweet T_i, we can define its text attribute as W_i and a term in that text as t_{W_i}. The term-frequency of a term can be calculated as such:

$$TF(t_{W_i}) = \frac{freq(t_{W_i})}{N} \tag{2.8}$$

where $freq(t_{W_i})$ is the absolute frequency of the term t_{W_i} and N is the size of the text. The *tf-idf* of a term can be calculted by the following formula:

$$TFIDF = TF(t_{W_i}) \cdot IDF(t_{W_i}) \tag{2.9}$$

where $TF(t_{W_i})$ is the term-frequency of term t_{W_i} and $IDF(t_{W_i})$ is the inverse document frequency of term t_{W_i}. The inverse document frequency is a statistic that reflects the importance of a term relative to a collection of text documents. It is given by the formula:

$$IDF(t_{W_i}) = log(\frac{ND}{DF(t_{W_i})}) \tag{2.10}$$

where ND is the number of documents in the collection and $DF(t_{W_i})$ is the number of documents that term t_{W_i} appears in.

Let x and y be two term-frequency vectors. To calculate the cosine similarity of those vectors we do:

$$sim(x,y) = \frac{x \cdot y}{\|x\| \|y\|} \tag{2.11}$$

with $\|x\|$ and $\|y\|$ being the norm of x and y, respectively. This measure calculates the cosine of the angle between the two vectors. The closer the angle is to 90, the more unlikely are they to be a match. The value obtained will be between 0 and 1, 0 meaning that the documents have no matching terms, and 1 meaning the documents have exactly the same terms. When attributes of the vectors evaluted are binary-valued (when it is only considered whether a word appears or not in a text) the cosine similarity function can be interpreted in terms of shared attributes. A variation of the cosine similarity for the described situation is the Tanimoto coefficient:

$$sim(x,y) = \frac{x \cdot y}{x \cdot x + y \cdot y - x \cdot y} \tag{2.12}$$

This formula gives us the ratio of the number of shared attributes between x and y to the total number of attributes.

2.3 TweeProfiles

In this section we will discuss Twitter's characteristics and also present the project that is the base of the work of this dissertation, TweeProfiles.

2.3.1 Twitter

Twitter is a social networking site and a micro blogging platform. It differs from its competitors in the perspective that users are allowed only posts with a maximum length of 140 characters. There are two types of relationships that can be established in Twitter. One can follow other users or in turn be followed by others users.

2.3.1.1 Concepts

There are important concepts, usually associated with a tweet that one should be aware of in order to have a better understanding of the social interactions within this network. Some of these concepts were introduced to social networking by Twitter or are unique to it:

Table 2.1: Twitter concepts

Concept	Description
Tweet	a post (or status update) in Twitter
ReTweet (RT)	share in your own feed a tweet from another user
Mention/reply (@<username>)	reference a user in your tweet
Hashtag (# <keyword>)	associate keywords to your tweet, that are related to its content (conversation theme, expressed feeling, among others)
Localization	attach the user's geo-coordinates to the tweet, at the time of posting

2.3.1.2 APIs

Twitter has two main APIs that allow developers to access its data: the REST API [Twi14b] and the Streaming API [Twi14c]. Both APIs require oAuth for authentication purposes. The main difference is that the REST API is request-based while the Streaming API is connection-based. In practice, this means that when using the REST API, one gets to query past information, being limited by the number of request allowed per user (in its current version, 15 or 180 requests are allowed per each 15 minute window, depending on the type of request); on the other hand, using the Streaming API one gets (almost) real-time data the only limitation being the volume of data one can get (1% of the total number of new tweets).

The other important difference between the two APIs is the type of information one can obtain. While using the REST API, one can get information about tweets (timelines, lists, favorites and search), users (friends and followers), locations (places and geo), direct messages and trends (trending topics), with the Streaming API there are only three types of streams that can be accessed: the public stream, which captures the tweets that are flowing through the network, and can be subjected to filters, to reduce the amount of information we get; the user stream, that gets all the information relative to one user; and the site stream, that also enables the same functionality as the user stream, but is meant for applications that need to get real-time information updates on many users at the same time.

2.3.2 Research using Twitter

Data mining reasearch with Twitter as a subject has been growing in the last years. The first work in this area was published in 2010 [BF10]. In this work an approach is described for doing sentiment analysis on Twitter by classifying tweets in two categories depending whether the texts

convey positive or negative feelings. This can be done manually anotating tweets (which is a very expensive and time-consuming task) or by counting the postive and negative occurrences on a text and then classifying a text according to the highest number of ocuurences. Examples of sentiment indicators are the presence of certain words ("good" and "awesome" or "bad" and "awful") or *smileys* (:) :(:D :P :O);

Sentiment analysis is also one of the goals of MOA-TweetReader [BHP11]. Developed on top of the MOA [BHKP10] framework, MOA-TweetReader [BHP11] is a tool that allows for the real-time processing of tweet streams. It connects with the Twitter StreamAPI and preprocesses the incoming tweets, applying a *tf-idf* filter and transforming them into sparse vectors of attributes or machine-learning instances, which are ready to be fed to the algorithms in MOA.

Considering works that apply clustering techniques to multi-dimensional Twitter data, in [BNG11] a tool is described that aims to identify real-world events through the mining of tweets. The framework consists of a clustering step and a classification step. Initially, clusters are formed considering temporal features (volume of messages in the timeframe of the event); social features (interactions of users in cluster's messages) and topical features (topics extracted from the messages). Then a classifier is trained to distinguish each cluster formed as an event or a non-event.

Finally, in [Lee12] a tool is described that performs multi-dimensional clustering (spatial, temporal and textual) over Twitter data. The aim of the tool is to detect events in real-time; an event is considered to be "a set of messages that are highly concentrated on some issues in a period of time", meaning that each cluster aims to represent a topic or group of topics that is being talked about at some point in time, in a certain place. It works as a two-pass system: the first pass is a text stream mining that extracts and clusters the tweets topics; the second pass clusters the results of the first pass according to their location. The result is a geo-spatial distribution of topics, in real-time, which can be mapped to a world map for visualization.

2.3.3 TweeProfiles

TweeProfiles [Cun13] is a data-mining tool that allows the extraction of multi-dimensional patterns from Twitter. The atttributes analyzed in each tweet are the geographical coordinates (spatial dimension), the timestamp (temporal dimension), the user (social dimension) and the text (textual/content dimension). It was developed as part of an M.Sc. thesis at the University of Porto and it is the basis of TweeProfiles2.

2.3.3.1 System Architecture

The architecture of TweeProfiles is shown in figure 2.3.

The data goes from the data sources to the server where it is pre-processed and then supplied to the clustering algorithm. The algorithm saves its results to a local database, where they can be accessed by the visualization tool (also developed as part of the same project), which is a webpage where the clusters are displayed in four different ways:

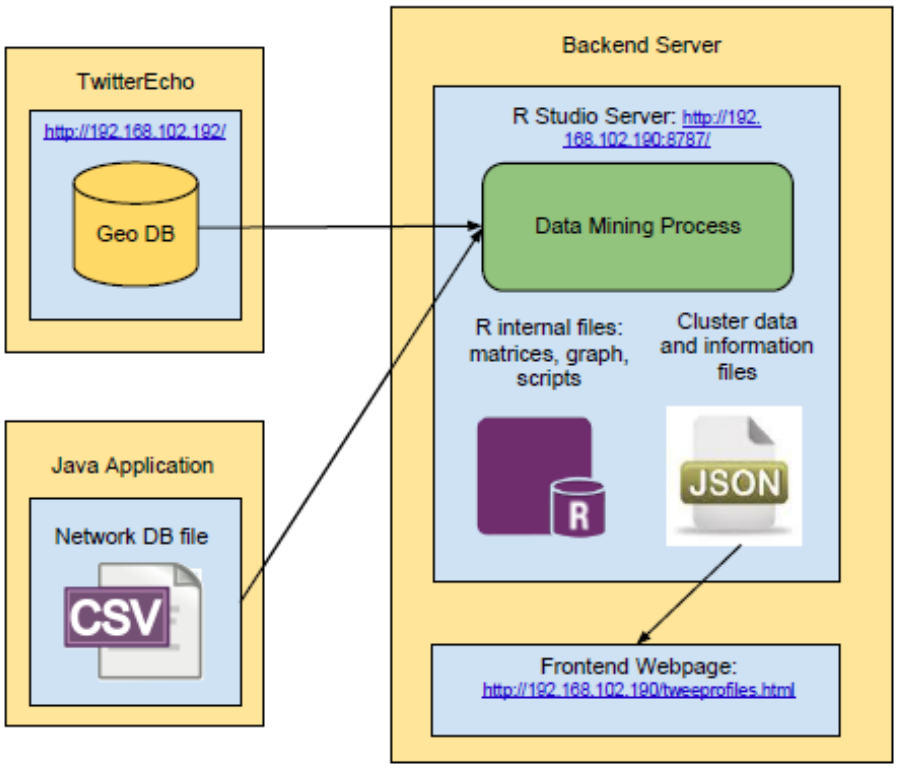

Figure 2.3: TweeProfiles system architecure

- a spatial visualization - the clusters are displayed as points in a worlds map, according to their coordinates;
- a temporal visualization - the clusters are displayed as bars starting and ending in the points corresponding to the dates of the oldest and newest point, respectively);
- a social visualization - the social network of the users in the cluster is displayed;
- a text visualization - the text contents of the cluster are displayed as a weighted word cloud;

2.3.3.2 Operation

The data mining process of TweeProfiles is based on dissimilarity matrices. Before the clustering process is put underway, 4 matrixes (one for each dimension) are calculated; these are $N * N$ matrices, where N is the total number of tweets in the database. Each element in the matrix represents a distance between two tweets, so if you consider the matrix M, $M_{i,j} = dist(i, j)$. The distance functions used, for each dimension are:

- **Spatial dimension** - Haversine distance

- **Temporal dimension** - Time difference ($t_1 - t_2$)
- **Social dimension** - Geodesic distance (nearest neighbour in the social graph)
- **Content dimension** - Cosine similarity (with TF-IDF)

After all the dissimilarity matrices (for every dimension) are calculated and normalized, the DBSCAN algorithm is applied to them, producing the final clustering results. These results can be recalculated giving different weights to each matrix, according o the desired analisys.

Chapter 3

TweeProfiles2

In this chapter we will explain the entire architecture and functioning of the developed tool, TweeProfiles2 and in the end compare it to its predecessor and discuss the changes.

3.1 Introduction

TweeProfiles2 was developed as an extension of TweeProfiles [Cun13] that allows clustering Twitter data streams on real-time. The clustering is done over three dimensions: the spatial dimension, the temporal dimension and the content dimension;

To analyze the spatial dimension we will use the tweet's geographical coordinates; this is an important restriction because not all of the tweets coming through the stream are geo-located.

The analyze the temporal dimension, we will use the tweet's timestamp. However, since the selected clustering algorithm already applies a temporal decay, we will instead try to find patterns among the hour and weekday of the posts.

Finally, to analyze the content dimension we will use the tweet's text. The setback here is that, because a tweet is limited to 140 characters, typically the number of words in each tweet is small and that may cause text similarities to be rather small.

This was done by adapting the DenStream algorithm in way that takes into account all of the mentioned dimensions, not only the numerical ones. The changes made will be explained in detail in this chapter.

3.2 Data Processing

The data that is passed on to the algorithm is extracted from tweets. However, looking at a JSON response from the Twitter API, there are over 30 fields in a tweet, some of them containing nested objects. Most of the information that comes in a API response is useless to our context and algorithm. So, the first pre-processing is applied in the data link step, where only some of the

fields are selected. Listing 3.1 shows the information stored about tweets after being stored for pre-processing.

```
{
    "tweetid":359419233295269888,
    "text":"Alguem fala como estao os protestos ?",
    "date":"2013-07-22T21:06:37.000Z",
    "lat":"-71.3553287",
    "lon":"-40.15760962"
}
```

Listing 3.1: Example tweet after selecting the fields required for our clustering approach

After this first pre-processing step, we still need to perform some data processing operations before passing the data to the micro-clustering algorithm. The first operation involves extracting the hour of the day and the day of the week from the date string. The range of values for these attributes are 0-23, for the hour and 1-7 for the weekday (Sunday through Monday).

After that, we need to process the text string, executing the following tasks, in this order:

1. Remove any URLs that may appear;
2. Remove all the punctuation from the text;
3. Detect the language of the text (using the *langdetect* library;
4. Tokenize the text;
5. Remove the stopwords from the text, depending on its language;
6. Apply a stemming algorithm to the remaining words, also dependent on the language.

The first four steps are pretty straightforward and self-explanatory. The last two steps are ordinary pre-processing tasks in text-mining projects, such as in [Kum12]. Stopword removal means removing words that are common to a language (for instance, prepositions or pronouns) and do not add any special meaning to the text. This is done by comparing the words in a text with a list of stopwords (that is why this task is language dependent). Stemming is the process of reducing a word to their *stem* or *root*. This is used, for instance, to group different tenses of the same verb. All the text pre-processing steps were done using the Lucene (see section 3.6.1).

Listing 3.2 shows how the tweet in listing 3.1 would look like after being pre-processed.

As we can see, beyond adding the aforementioned fields, we also removed the ones with the unique identifiers ("tweetid"). These fields are not needed in the clustering process as the tweetID does not give us any relevant information about the tweet.

From now on in this chapter, we will consider a tweet to be the tuple

$$TW(lat, lng, hou, wkd, date, \overline{txt}) \tag{3.1}$$

```
{
    "text":"fala est protest",
    "date":"2013-07-22T21:06:37.000Z",
    "hour": 21,
    "weekday": 2,
    "lat": -71.3553287,
    "lon": -40.15760962
}
```

Listing 3.2: Example tweet after pre-processing

In equation 3.2 *lat* and *lng* are the tweet's latitude and longitude coordinates, respectively, *hou* and *wkd* are the hour and weekday, $date$ is the date and $\overline{txt}$ is the vector of words of the tweet' text.

3.3 Distance Functions

The distance function we applied in our adaptadion of DenStream is a piece wise function, that is composed of the haversine function, the euclidean distance and the cosine similarity.

$$dist() = \begin{cases} haversine formula & \rightarrow TW[lat, lng] \\ euclidean formula & \rightarrow TW[hou, wkd] \\ cosine similarity & \rightarrow TW[\overline{txt}] \end{cases}$$

This function is used to calculate the distance between two tweets as well as the distance between a tweet and a cluster. In the latter case, the distance is calculated between the tweet and the cluster's center (see 3.4.1). Different attributes are used with different parts of the function:

- With the geographical coordinates (*lat* and *lng*), we use the Haversine formula, which gives the great-circle distance, in kilometers, between two points;
- With the hour and weekday attributes, we use the euclidean distance formula;
- With the texts term-frequency vectors, we calculate the cosine similarity between two texts;

However, as the order of magnitude between these functions is very disparate, we have to normalize them. As it is a distance value, it must always be greater than 0. As we already have one distance component (cosine similarity) that is in the range $[0, 1]$, if we apply a min-max normalization, we can obtain a similar scale in the other components. As all distance functions have minimum 0, we only need to divide by the corresponding maximum values. The maximum values for each formula, in our project's context, is:

Table 3.1: Maximum distance values

Formula	**Tweet fields**	**Maximum**
Haversine Formula	latitude, longitude	20.020 (km)
Euclidean Formula	hour, weekday	$\sqrt{565} = 23,77$
Cosine Similarity	text	1

Also, in all the distances a value of 0 means that the point is the closest possible (maximum similarity), whereas in the cosine similarity formula, a value of 0 means that the texts are totally different. So we also have to convert the result of the cosine similarity piece of the function, by subtracting it to 1.

The obvious advantage of having a piecewise distance function, is that we can tinker with the weights of each component, which can output different results for the same dataset (or data stream). The chapter 4 we will develop more on this subject.

3.4 Clustering

As there was no algorithm that combines stream clustering of numerical, categorical and textual attributes, we had to develop one. We used DenStream (see 2.1.1.3) as a framework, adding the

necessary mechanisms to allow it to perform the task at hand. We chose DenStream for several reasons:

1. The macro clustering algorithm used in the DenStream framework is DBSCAN, the same algorithm used in TweeProfiles
2. DenStream is already implemented in MOA, a proven stream data mining framework
3. DenStream is based on the Euclidean distance. However, it is generic enough to allow other distance formulas to be used with the same clustering process

We called our adapted algorithm HybridDenStream, as it is able to cluster data points with different types of information, such as tweets.

Before we could adapt DenStream to cluster these three dimensions, we also had to adapt the underlying micro-cluster structures so they could summarize information of several types. So, we will first specify our hybrid micro-cluster structures and then explain the functioning of HybridDenStream.

3.4.1 Hybrid MicroCluster

As explained in 2.1.1.3, DenStream is a stream clustering algorithm with two clustering steps: online *micro-clustering* and offline *macro-clustering*.

The micro-clustering step is meant to summarize the information coming through the stream, creating small clusters (micro-clusters) that are stored and later used in the macro clustering step in order to generate the final clustering results.

In order to process tweets' information, besides altering the DenStream default distance function, we also had to change the micro-clusters' structure, so we are able to summarize all the information concerning tweets that we need for the clustering process So we created a HybridMicroCluster structure, that is meant to represent a small aglomeration of tweets. We can also think of it simply as a "big tweet".

A hybrid-micro-cluster, at time t is defined as $HMC(w, ww, c, r)$ for a group of similar tweets (as defined in equation 3.2) $TW_1 \dots TW_n$ with time stamps $T_1 \dots T_n$, and text vectors $W_1 \dots W_m$:

$$w = \sum_{j=1}^{n} f(t - T_j), w > \mu \text{ is the weight;} \tag{3.2}$$

$$c = \begin{cases} \frac{\sum_{j=1}^{5} f(t-T_i) TW_{i_j}}{w} \\ \frac{\sum_{k=1}^{m} f(t-T_i) W_k}{ww} \end{cases} \text{ is the center;} \tag{3.3}$$

$$r = \frac{\sum_{j=1}^{n} f(t - T_j) dist(X_j, c)}{w}, r \leq \varepsilon \text{ is the radius;} \tag{3.4}$$

where $f(t)$ is the temporal decay function (as defined in equation 2.1.1.3 and $dist(X_{i_j}, c)$ denotes the composed distance function between the point X_i and the center c.

Like the original DenStream's micro-clusters, hybrid micro-clusters also are characterized by a weight, a center and a radius, with the weight and radius calculated in the same way. However, the concept of center in a data point with categorical and textual attributes is quite abstract. So we consider the center of a hybrid micro cluster to contain not only its numerical attributes (in this instance, the sums of the coordinates, weekday and hour values divided by the weight of the cluster) but also a term-frequency vector (containing the words of the clustered tweets paired with their respective relative frequencies).

In the original DenStream, the summary information for each cluster is stored in two d-dimensional vectors; these vectors contain the linear and squared sums of the points in the cluster, for each dimension. While this information may be sufficient to compute the cluster's radius *a-posteriori* using only the euclidean distance, it is not enough to do so for our custom distance function. So, instead of keeping the squared sums for each (numeric) attribute, we keep the sums of the distances of each point in the cluster to its center.

3.4.2 Clustering mechanism

We will now explain how a new point is processed in HybridDenStream which is the same way as in regular DenStream (see section 2.1.1.3):

1. When a new point X arrives, the nearest potential-micro-cluster, a, is identified and we try to add the new point to it;

2. If a (with the new point added) violates the radius constraint ($r < \varepsilon$), we remove X from it, identify the nearest outlier-micro-cluster, b, and add X to it;

3. If b respects the radius constraint, we check to see if the weight of b is enough to transform it into a PMC ($w > \beta * \mu$). If it is, we remove b from the OMC buffer and add it to the PMC buffer;

4. Finally, if b violates the radius constraint, we create a new OMC c with only a point X and add it to the OMC buffer.

To initialize the algorithm, the first 1000 points are clustered using DBSCAN, meaning that clusters will be formed if the distance between two points is less than the parameter ε. The algorithm's parameters are explained in more detail in table 4.3.

The biggest difference between the original DenStream and Hybrid DenStream is the way in which the distance between individual points and clusters as well as between clusters is calculated. In DenStream, as it is only prepared to deal with numerical attributes, all the distances are calculated with the Euclidean formula (see section 2.2.1). In Hybrid DenStream, as our attributes are of different data types, our distance function is composed of several components, as defined in section 3.3

After the resulting hybrid micro-clusters are produced, they are stored in a local database, where they can later be accessed by the macro-clustering part of the framework. While the micro-clustering step is always running (therefore it is called online-step), the macro-clustering step works on-demand, meaning that it only operates when a clustering request arrives.

The algorithm used to generate the macro-clusters is DBSCAN. It simply calculates the distance between the micro-clusters' centers and if it is less than the ε parameter, it adds them. Like DenStream's micro-cluster structures, hybrid micro-cluster mantain the additivity property.

3.5 Visualization

A simple visualization tool was developed to help analyze and make sense of the obtained results. It allows three types of cluster visualizations: a spatial visualization, a temporal visualization and a content visualization. The tool works by loading the (micro-)clustering results from a MySQL database and loading each clusters' information according to the visualization type.

The spatial visualization, as shown in figure 3.1, consists of a world map where the red dots represent micro-clusters; the clusters are represented by a blue circular area, not visible in the picture. The clusters are plotted according to their center's geographical coordinates.

Figure 3.1: Spatial visualization

The temporal visualization, as shown in figure 3.2, consists of a *x*-*y* graph where the *x* axis represents the day of the week (Sunday through Saturday) and the *y* axis represents the hour. The clusters are represented by bubble and are plotted according to their center's hour and weekday

values, which define the clusters center in the graph; the radius of the spheres represents the number of points (tweets) in that cluster.

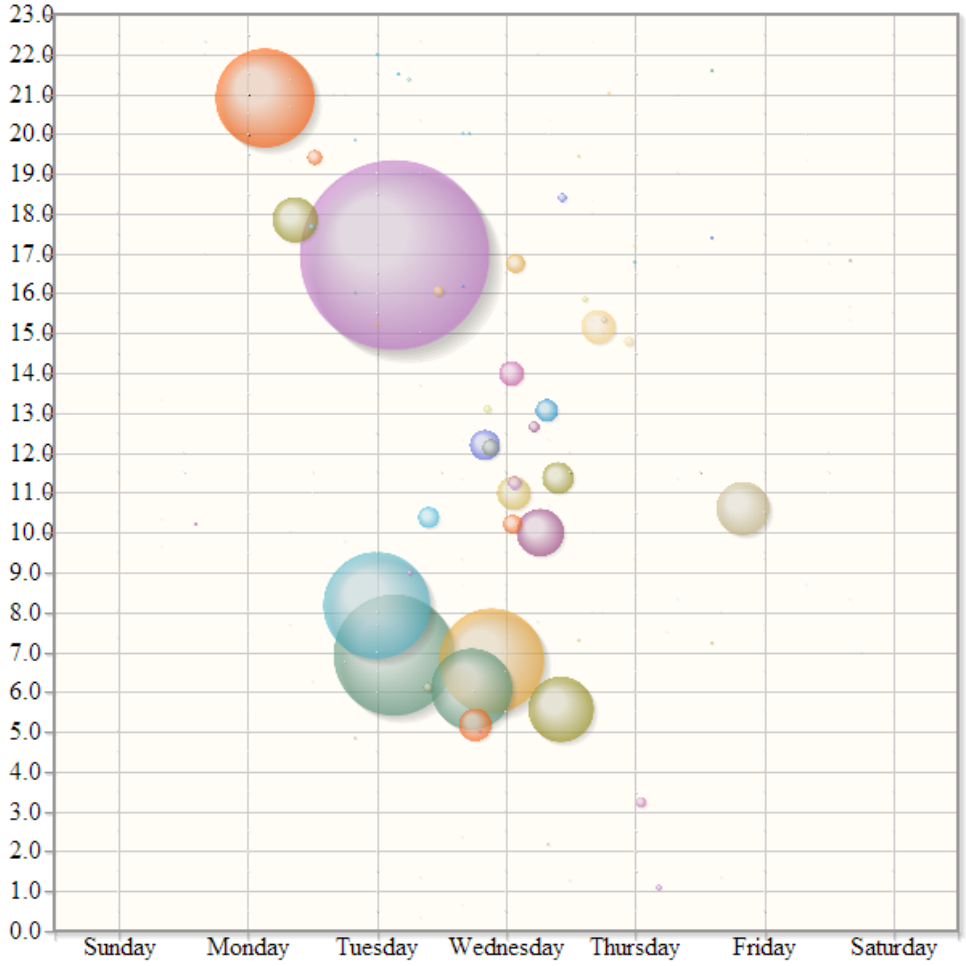

Figure 3.2: Temporal visualization

The content visualization, as shown in figure 3.3, consists of a word cloud, where the size of each word represents its frequency. This means that the bigger the word is, the more times it appears in the texts of the tweets in the cluster.

Figure 3.3: Content visualization

3.6 System Architecure

The architecture of the proposed solution is described in the following picture:

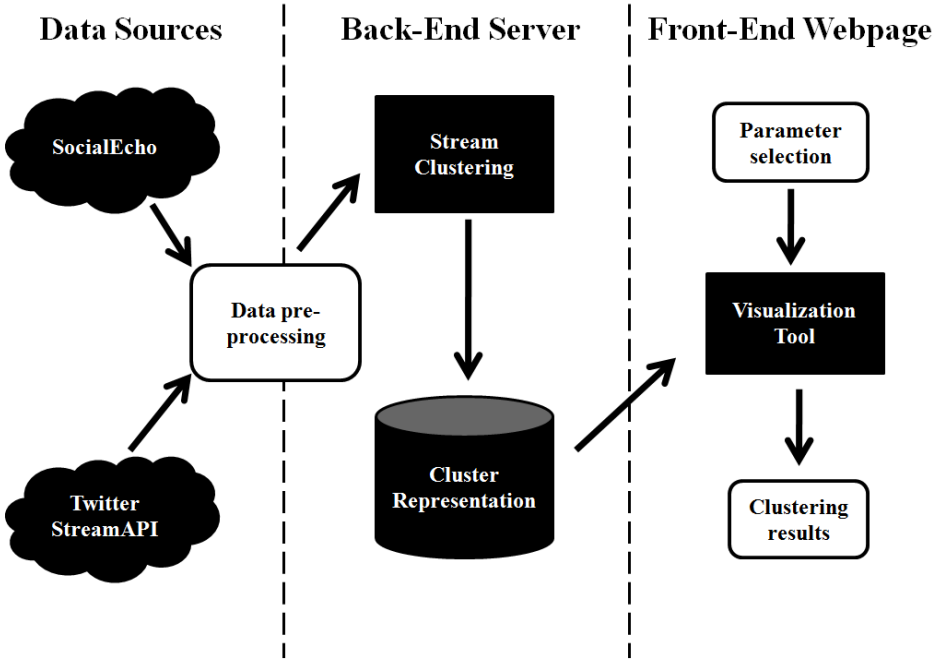

Figure 3.4: TweeProfiles2 high-level architecure

The data stream will be pipelined to a back-end server, where it will feed the micro-clustering algorithm, passing through several data pre-processing steps (see sect 3.2). The resulting micro-clusters will then be stored in a relational database, where they will accessible to the macro-clustering algorithm, whenever a clustering request arrives. The resulting clusters are then passed to the visualization module, where they will be displayed for analysis of the results.

3.6.1 Harnessed Tecnologies

TweeProfiles2 is developed entirely using the Java language for all the algorithm and data processing tasks, with a background MySQL database used to store the micro-clustering results. Several Java external libraries are used to perform some tasks along the entire KDD process. These libraries are described in table 3.2.

Table 3.2: Java libraries

Name	Reference(s)	Description
MOA/Weka	[BHKP10]	MOA is a data stream mining framework which has a collection of several algorithms (including DenStream and DBSCAN) and evaluation methods. It works on top of WEKA, one of the most popular Java data mining frameworks
Apache Lucene	[Apa14]	Lucene is a text search engine library. It offers several text processing methods, including methods for stemming and stopword removal in texts, which are required in TweeProfiles2
org.json	[JSO14]	Library that allows for the processing of JSON objects, which is the language used in the responses of the Twitter API's
langdetect	[Cod14]	Library used for detecting the language in which a text is written. Can detect up to 53 languages with a precision of 99%
MySQL connector	[MyS14]	Driver for working with MySQL databases in Java

Chapter 4

Results

In this chapter we present the test dataset and testing setup and explore the results obtained in the tests carried out.

4.1 Exploratory Data Analysis

To test our algorithm, we used a dataset with 50,735 geo-located tweets from SocialBus (formerly TwitterEcho [BOM+12]), from June to August of 2013. Most of these tweets are related to the social uprising that occured in Brazil in 2013. The languague of the tweets, according to the *langdetect* library are distributed as such:

Table 4.1: Test dataset languages

Language	**Number of tweets**
Romanian	615
Turkish	177
Lithuanian	120
German	243
Finnish	158
French	133
Slovene	507
Italian	1249
Portuguese	43582
English	306
Spanish	1255
Unclassified	50
Other languages	1350
Total	**50735**

As we can see in table 4.1, Portuguese is by far the language with the most tweets in the test dataset. So, to reduce the noise and confusion in the textual dimension, in all the testing setups, only the tweets that were detected to be in portuguese were used.

In figure 4.1 we have the geographical distribution of the test dataset. We can clearly identify two major "clusters" of tweets, in South America (Brazil mainly) and Europe, and an area with subtancial density in North and Central America.

Figure 4.1: Test dataset geographical distribution

The hourly distribution of the test dataset if shown in 4.2. As we can see, the activity peaks during the night hours, with the most tweets being created around midnight. All times are in GMT (Greenwich Mean Time).

In figure 4.3 we have the day of the week distribution of the test dataset. The peak day is Thursday, and we can see the majority of the activity being at the end of the week (Thursday, Friday and Saturday).

Finally, in figure 4.4, we can see the dates of creation of the tweets. As we can see, almost 80% of the total tweets are from June 2013, and only a residual sample is from August 2013.

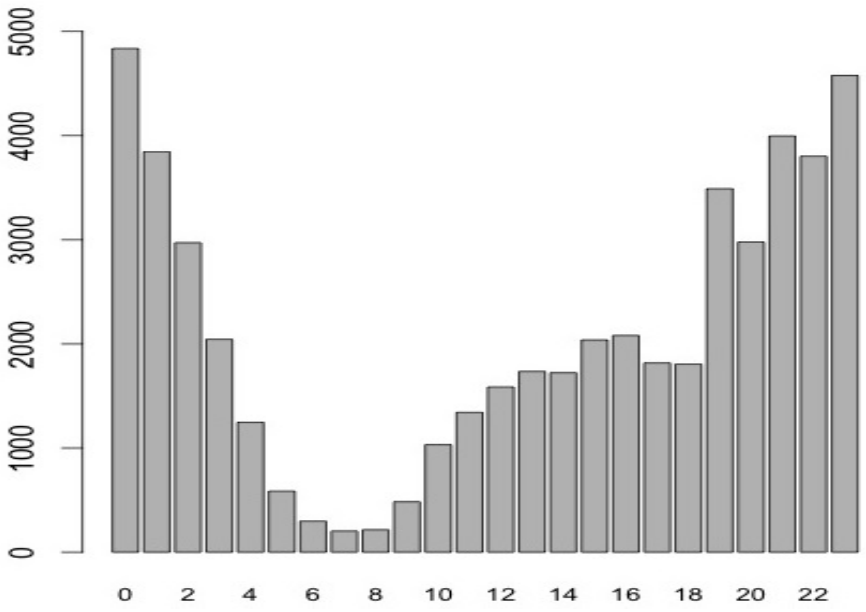

Figure 4.2: Test dataset hour distribution

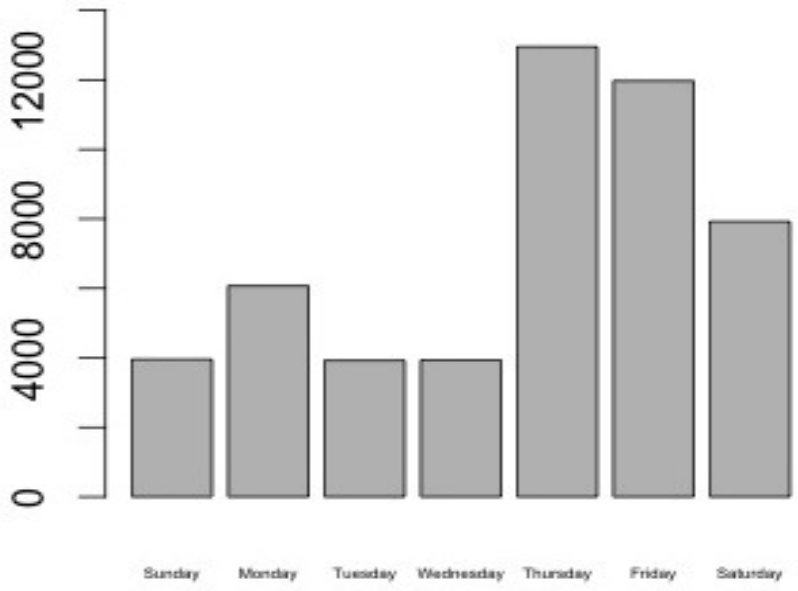

Figure 4.3: Test dataset weekday distribution

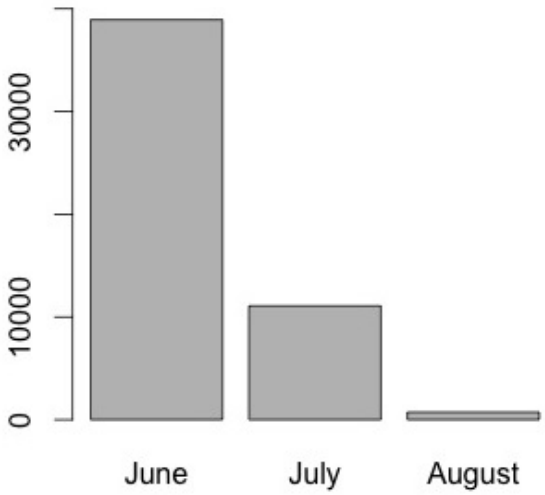

Figure 4.4: Test dataset date distribution

4.2 Streamalizer

Streamalizer is a simple tool developed to help in the testing of the stream clustering algorithm used. It takes any static dataset (local or remote files or databases) as input and outputs it as a stream, with the arriving times of the points following a normal distribution. It was integrated with the SocialBus database with a data filter, meaning we could retrieve tweets from different time periods. However it is generic enough to be easily integrated with other platforms.

4.3 Testing setup

To assess the algorithm, we designed a series of testing rounds, each with a different set of conditions. Table 4.2 shows the components of the algorithm's distance function in each of the testing rounds

Table 4.2: Testing rounds distance combinations specification

Combination	Spatial Component	Temporal Component	Textual Component
DC1	100%	0%	0%
DC2	0%	100%	0%
DC3	0%	0%	100%
DC4	33%	33%	33%

Besides the distance function weighting, HybridDenStream also has a set of parameters that can be altered to produce different clustering results. These parameters are explained in table 4.3:

The offline-step uses DBSCAN, which also has two input parameters, epsilon and MinPoints, which have the same meaning as the ones referred in table 4.3, but instead are used in the macro-clustering step of the process.

Table 4.3: Hybrid DenStream parameters

Name	Abbrev	Description	Min Value	Max Value	Default Value
Epsilon	eps	Defines the minimum radius of a HMC	0	1	0.1
MinPoints	mp	Defines the minimum number of points in the ε-neighbourhood to create a hmc (also used as μ parameter)	1	∞	2
InitPoints	ip	Number of points for initialization	100	∞	1000
μ	μ	used in the PMC/OMC restriction	1	∞	1
Beta	β	used in the PMC/OMC restriction	0	1	0.2
Lambda	λ	Used in the time decay function; affects the decay rate of the stream	0	1	0.25
Processing speed	s	Defines the number of instances (tweets) per time unit	1	∞	100

4.4 Results

Throughout the results discussed below, we will perform testing rounds with the setups described in table 4.2. Also, in each setup, we will perform a sensitivity analysis of the input parameters of the algorithm, trying to discover the optimal values for each parameter.

4.4.1 Spatial dimension

The results of this testing round were obtained using table's 4.2 DC1 parameters, meaning that to form clusters, only the spatial dimension is taken into account to calculate the distances, both in the micro and macro-clustering step.

4.4.1.1 Testing rounds and sensitivity analysis

We performed several rounds of testing with the following parameters of DenStream and DBSCAN, as shown in table 4.4.

Test 1.1 and 1.2 did not produce satisfatory results (few micro-clusters with a lot of instances) suggesting that the ε parameter for this dimension should be a little lower. So, in tests 1.3 and 1.4, we lowered the epsilon parameter a the results improved significantly. Test 1.5 was done to assess the importance of DBSCAN minPts; for this particular setup, as test 1.3, which has the same ε values but a higher minPts value.

Table 4.4: Testing round 1

Tests	DenStream		DBSCAN		Clusters	
	eps	minPts	eps	minPts	Micro	Macro
1.1	0.1	1	0.2	4	1	0
1.2	0.05	1	0.1	4	16	1
1.3	0.01	1	0.02	4	187	11
1.4	0.005	1	0.01	2	156	9
1.5	0.01	1	0.02	2	187	11

4.4.1.2 Results and discussion

In figures 4.5 and 4.6 we can see the results (micro and macro, respectively) of a test of the spatial dimension. Each red dot in figure 4.5 represents a micro-cluster; in this representation all the dots have the same size regardless of its micro cluster's characteristics. Each blue circle in figure 4.6 represents a cluster. The radius around each cluster is not the actual radius of the cluster but instead is a value calculated, based on the cluster's radius and number of points.

Each time point is a moment in the simulation where we stored the micro and macro clustering results at that time. This is useful to see the evolution of the clusters during the stream. In figure 4.6 we can cleary see the clusters growing in size through the four time points. The resulting clusters can be deemed adequate, because as we saw before (figure 4.1, the majority of the tweets from our dataset come from South America (Brazil).

4.4.2 Temporal dimension

The results of this testing round were obtained using table's 4.2 DC2 parameters, meaning that to form clusters, only the temporal dimension is taken into account to calculate distances, both in the micro and macro-clustering step.

4.4.2.1 Testing rounds and sensitivity analysis

We performed several rounds of testing with the following parameters of DenStream and DBSCAN, as shown in table 4.5.

Table 4.5: Testing round 2

Tests	DenStream		DBSCAN		Clusters	
	eps	minPts	eps	minPts	Micro	Macro
2.1	0.1	1	0.2	4	139	1
2.2	0.1	1	0.2	2	139	1
2.3	0.01	1	0.02	4	0	0
2.4	0.1	1	0.1	4	139	10
2.5	0.1	1	0.1	2	139	10

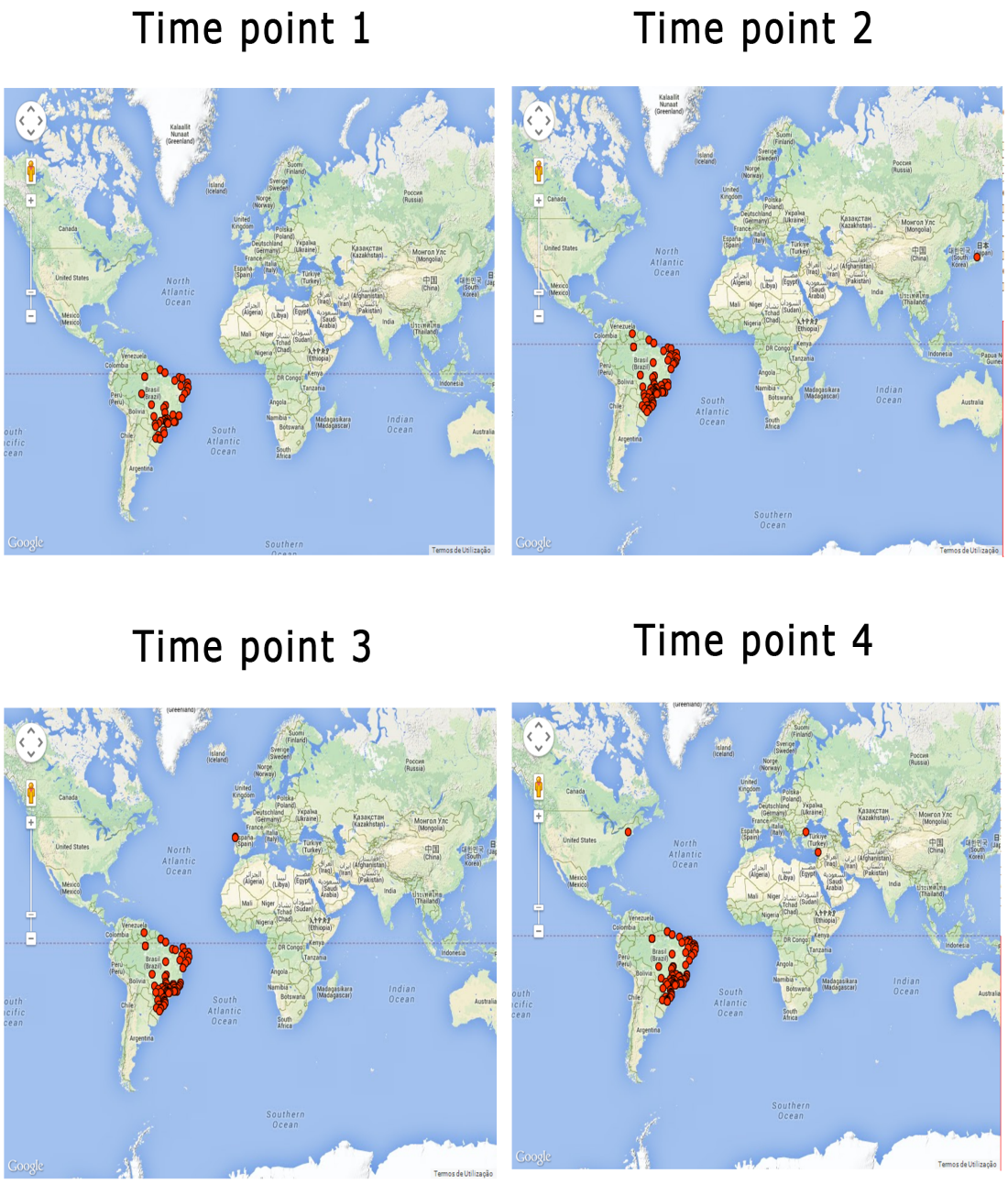

Figure 4.5: Spatial micro-clustering results

As shown above, the ε parameter has a high sensitivity to this dimension. The best results were achieve with a value of 0.1 for both epsilon parameters. Also, like in the previous tests, we can conclude that the DBSCAN minPts parameter does not have a large effect in the final clustering results.

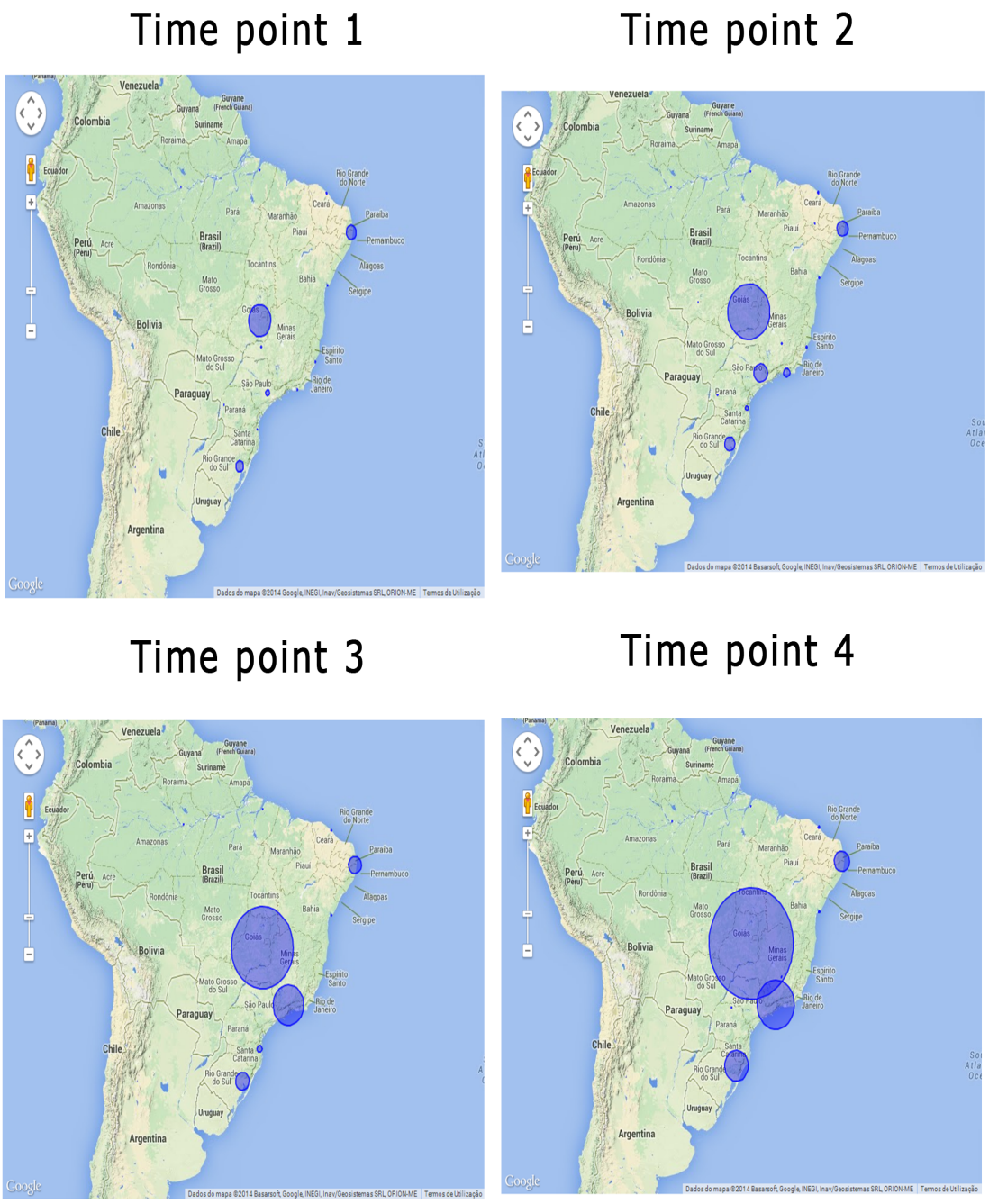

Figure 4.6: Spatial clustering results

4.4.2.2 Results and discussion

In figures 4.7 and 4.8 we can see the results (micro and macro, respectively) of a test of the temporal dimension. Each sphere in the figures represents a cluster over the two temporal dimensions (hour and weekday); the radius of the sphere is calculated from the number of points in the clusters,

so a bigger radius represents a cluster with higher number of points.

Each time point is a moment in the simulation where we stored the micro and macro clustering results at that time. This is useful to see the evolution of the clusters during the stream. In figure 4.7 we can see a big micro-cluster being formed, decaying over time; later in the stream, we can see new, smaller micro-clusters appearing. This can be explained if the dataset has a high concentration of tweets in a single day, and the rest of the days have an even distribution of tweet. We saw in figure 4.4 that about 80% of the tweets in the dataset are from a single month, so it is possible that a large part of those tweets are from the same day. in In figure 4.8, we see that in time point 1 the algorithm was unable to find any clusters in the stream; in time point 2 we see a strange thing: the cluster that is formed is rather distant from the micro-cluster shown in time point 2 of figure 4.7; this can be explained by the fact that the *minPts* parameter for DBSCAN is 2 (meaning that a clusters has to have a minimum of 2 micro-clusters), and for that reason the early big micro-cluster shown in figure 4.7 is ignored and instead a cluster is formed with smaller micro-clusters in that region.

4.4.3 Content dimension

The results of this testing round were obtained using table's 4.2 DC3 parameters, meaning that to form clusters, only the content dimension is taken into account to calculate distances, both in the micro and macro-clustering step.

4.4.3.1 Tests and sensitivity analysis

We performed several rounds of testing with the following parameters of DenSream and DBSCAN, as shown in table 4.6.

Table 4.6: Testing round 3

Tests	DenStream		DBSCAN		Clusters	
	eps	minPts	eps	minPts	Micro	Macro
3.1	0.1	1	0.2	4	12	0
3.2	0.3	1	0.6	4	18	0
3.3	0.3	1	0.6	2	18	2
3.4	0.4	1	0.6	4	40	1
3.5	0.4	1	0.6	2	40	3

The results of this testing round are not as satisfactory as the previous ones. This can be attributed to the fact that, as the compared texts are small, the computed similarity is also typically small. The ε values for this testing round were the highest, and still the number resulting clusters is the smallest of all testing rounds. However, if we increase this value even more, the quality of the results may decrease, because the algorithm will merge text instances that have very little similarity.

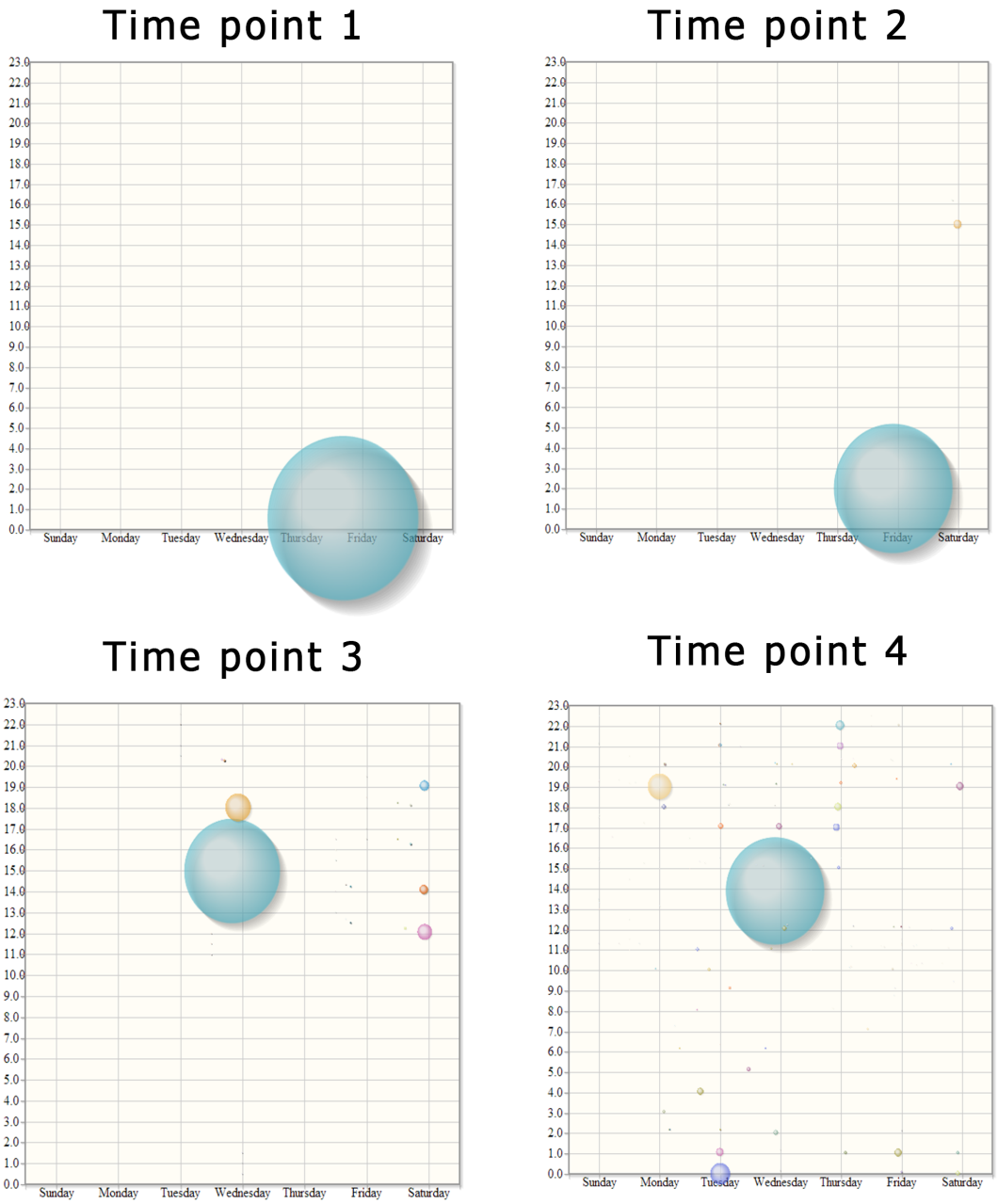

Figure 4.7: Temporal micro-clustering results

4.4.3.2 Results and discussion

In figure 4.9 we can see the results of a test of the content dimension. It shows, for each time point, the most frequent words in the clusters formed at that time. The sizes of the words are proportional to their frequency relative to others. We can see that although the words alter their

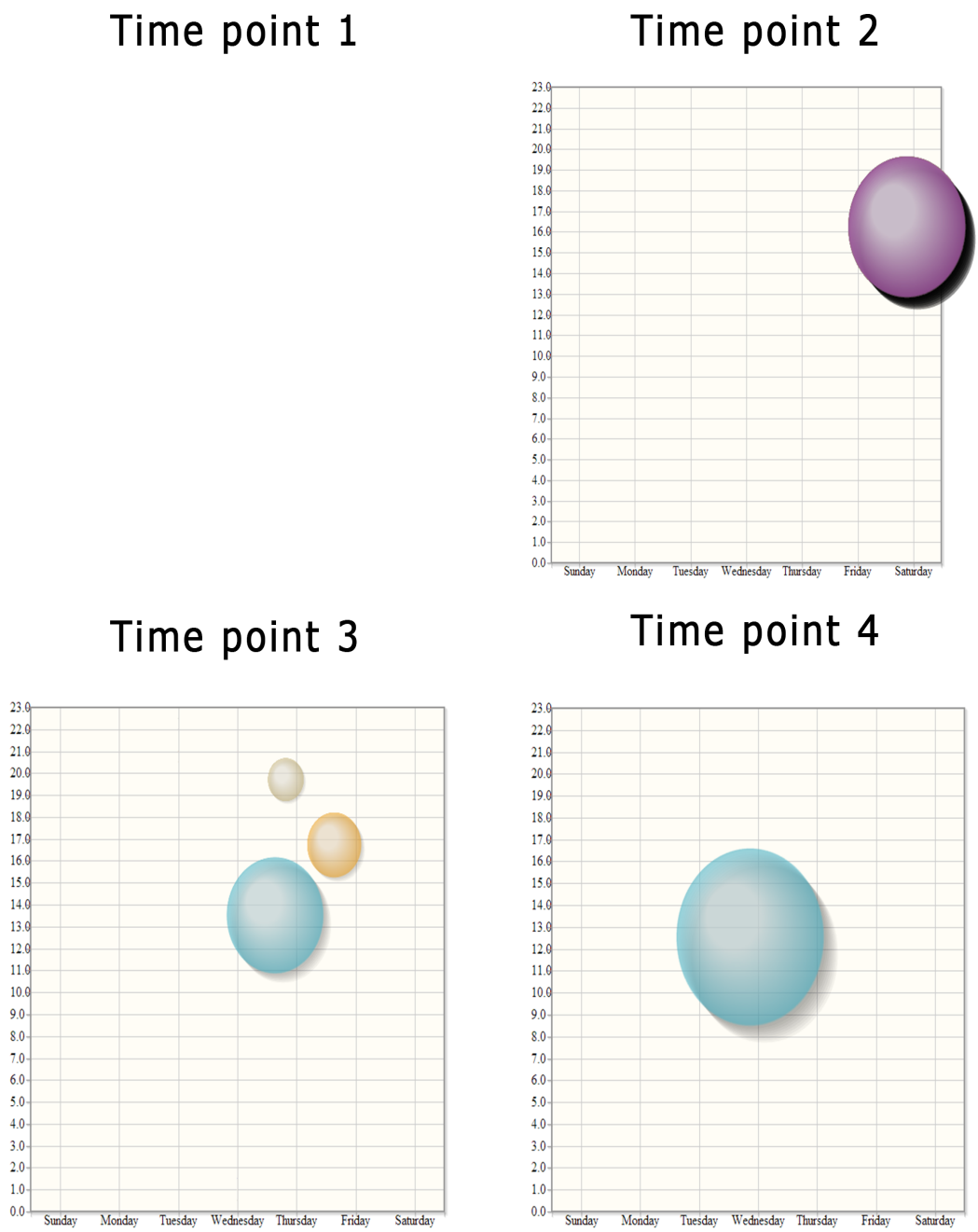

Figure 4.8: Temporal clustering results

relative importance thoughout the stream, the most frequent words remain consistent during the whole experiment. Most of these words, like "vemprarua" ou "protestos" reflect the social uprising that was felt in Brazil during that period.

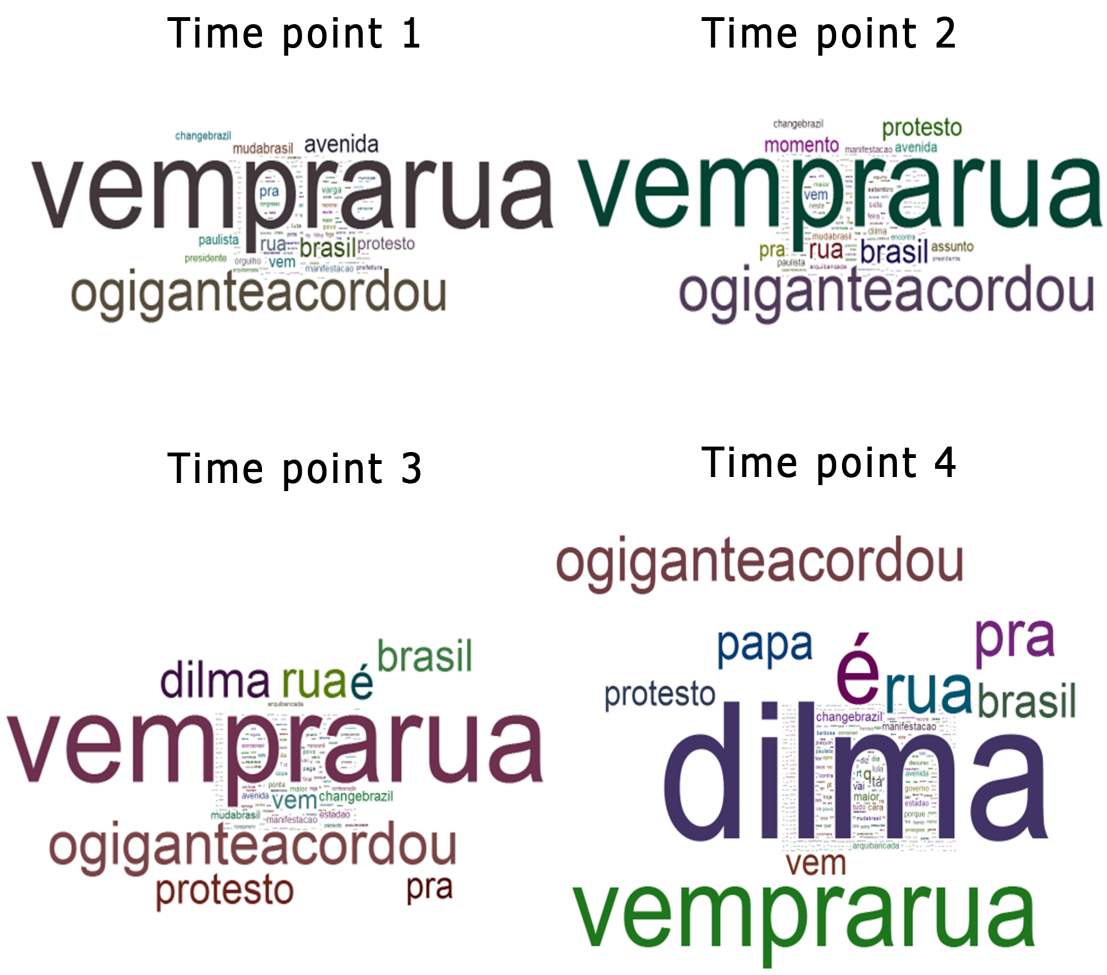

Figure 4.9: Content clustering results

4.4.4 All dimensions

The results of this testing round were obtained using table's 4.2 DC4 parameters, meaning that to form clusters, all the dimensions are taken into account to calculate distances, both in the micro and macro-clustering step.

4.4.4.1 Testing rounds and sensitivity analysis

We performed several rounds of testing with the following parameters of DenSream and DBSCAN, as shown in table 4.7.

As expected, when combining all the dimensions, we had to increase the epsilon values significantly. Even so, of the 625 MC produced in tests 3.4 and 3.5, only a small part of them was used to produce the final clusters (10 and 26, respectively). As in the previous testing round, we could not increase epsilon much more or we would risk having resulting clusters with no real meaning

Table 4.7: Testing round 4

Tests	DenStream		DBSCAN		Clusters	
	eps	minPts	eps	minPts	Micro	Macro
4.1	0.1	1	0.2	4	0	0
4.2	0.3	1	0.6	4	2	0
4.3	0.4	1	0.6	2	30	2
4.4	0.5	1	0.6	4	625	8
4.5	0.5	1	0.6	2	625	14
4.6	0.5	1	0.75	4	625	3
4.7	0.5	1	0.75	2	625	28

in relation to the dataset. The best results were achieved in test 4.4 and 4.5, because, as the epsilon constraints were relaxed, more points were included in the results.

4.4.4.2 Results and discussion

In figures 4.10, 4.11, 4.12, 4.13 and 4.14 we can see representations, similar to those above, of the micro and macro clustering results of a test done over all the aforementioned dimensions.

As discussed before, the epsilon parameters for this test had to be slightly higher than the other tests. This caused an impact in the macro-clustering step, where a small number of clusters was formed, but not so much in the micro-clustering step, which produced a considerable amount of clusters with a fine level of granularity. If we compare each of the micro-clustering temporal and spacial representations with the ones in the previous tests, we can see more micro-clusters in the former ones than in the latter.

As for the content clustering, the results of this test and the previous one are very similar, proving that, although not very complex, the text mining component of the algorithm maintains its consistency though the tests.

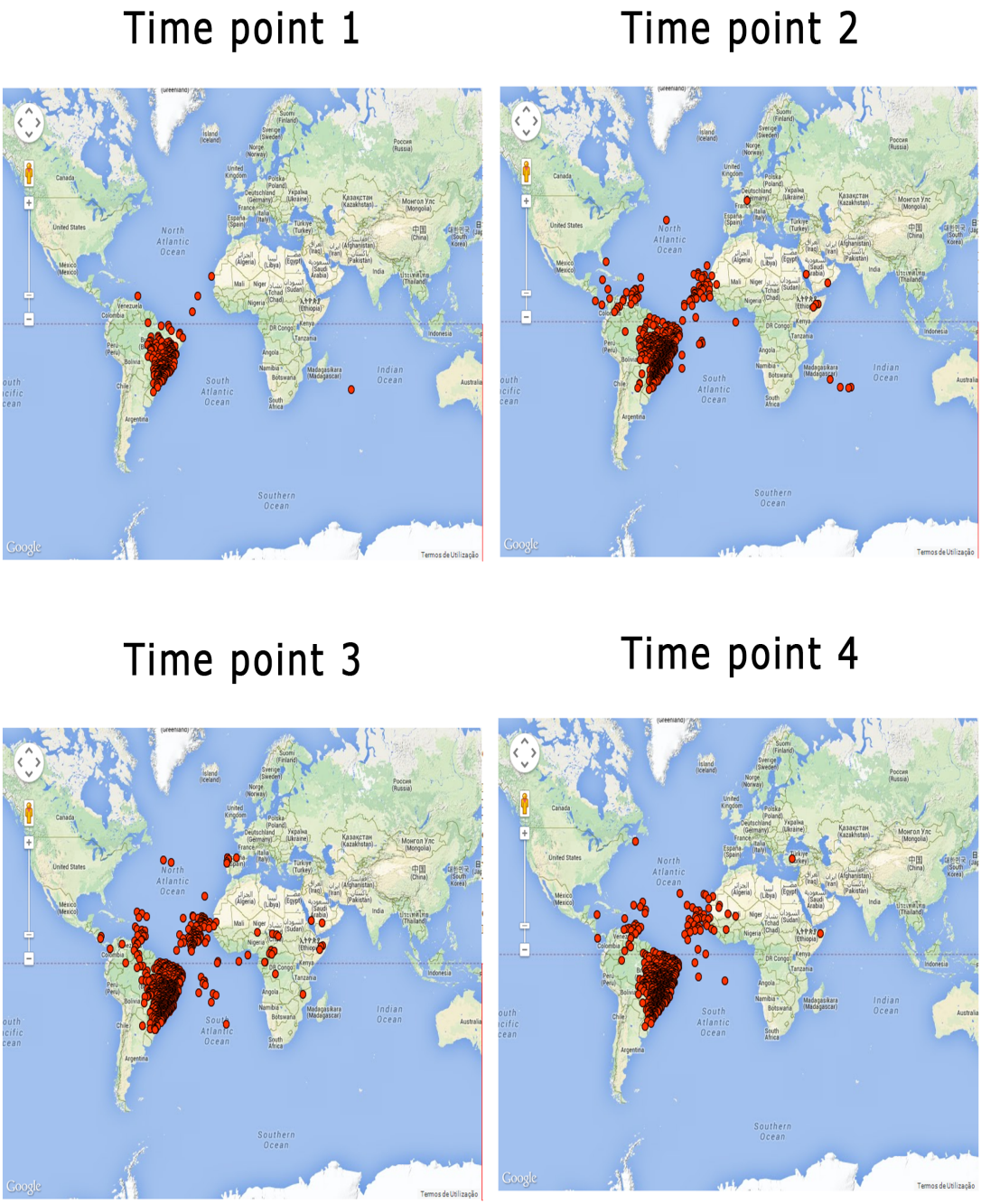

Figure 4.10: Multi-dimensional micro-clustering results (map view)

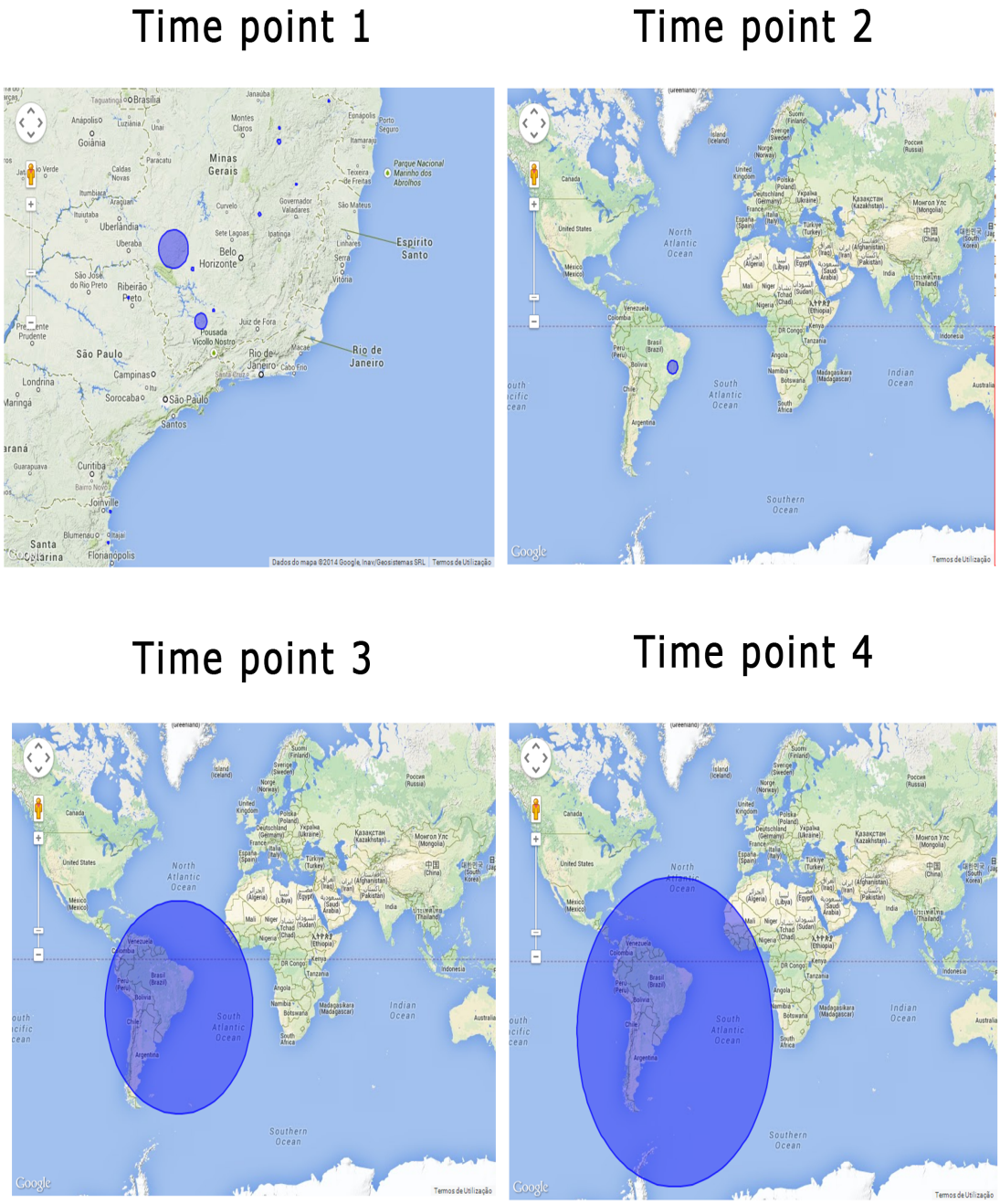

Figure 4.11: Multi-dimensional clustering results (map view)

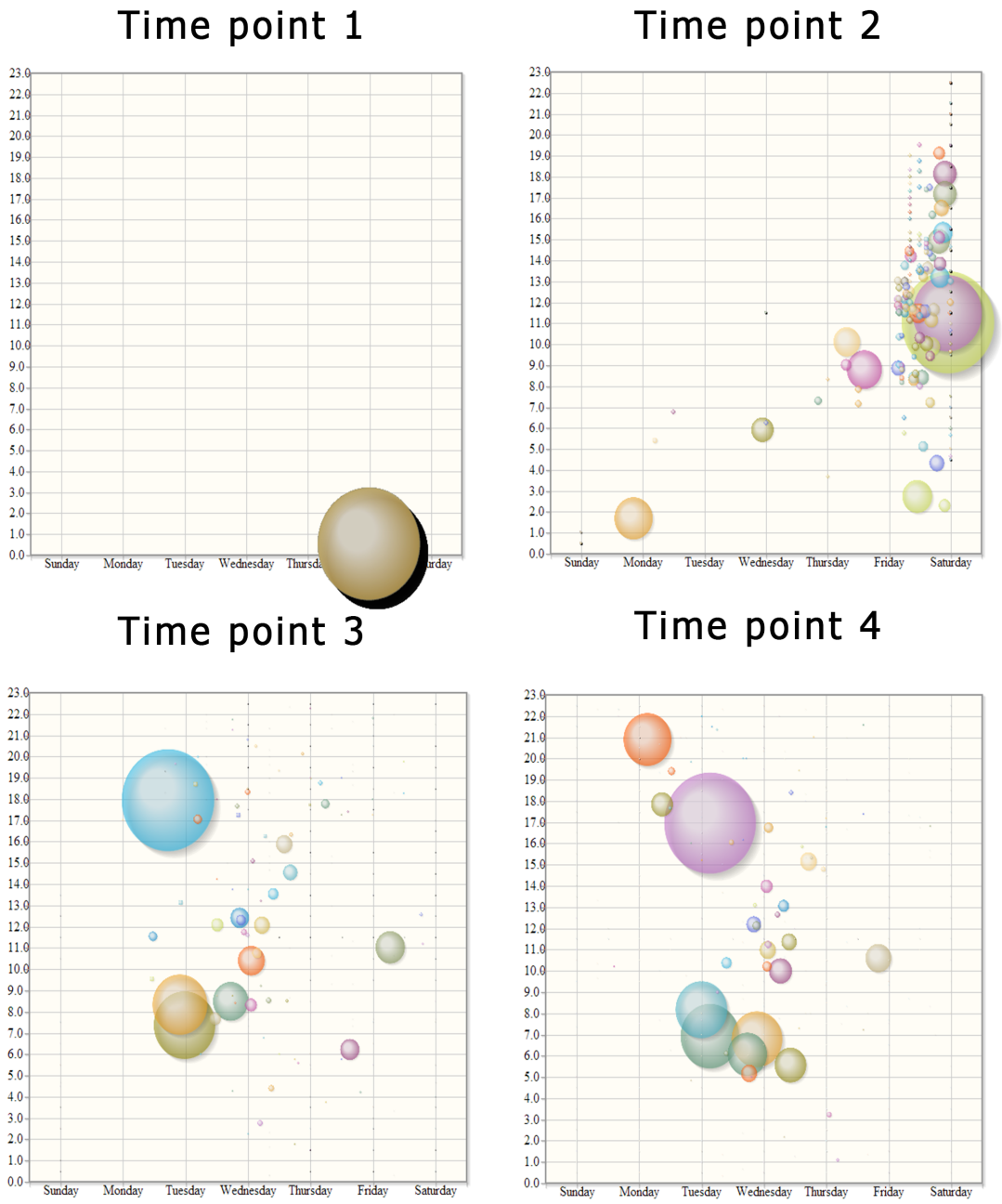

Figure 4.12: Multi-dimensional micro-clustering results (time view)

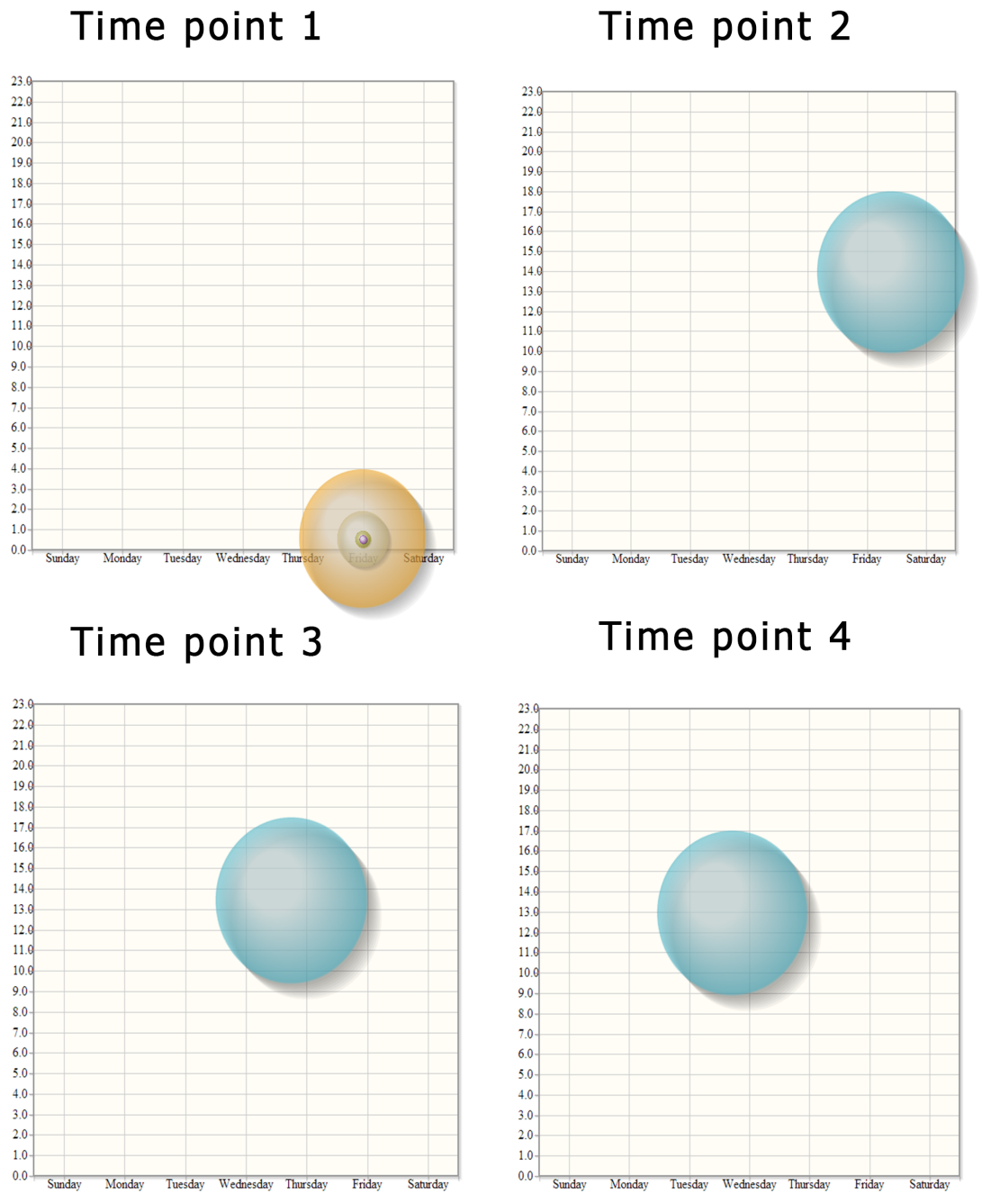

Figure 4.13: Multi-dimensional clustering results (time view)

Figure 4.14: Multi-dimensional clustering results (words view)

Chapter 5

Conclusions and Future Work

In this chapter we will talk about the conclusions achieved with the development of this project, as well as the future work that can be done to improve it

5.1 Summary

The objective of this work was to produce a tool that allowed real-time, multi-dimensional clustering of a Twitter stream. In order to achieve that, we developed a data-mining process:

1. **Data Extraction** - using Streamalizer[4.2] to connect to SocialBus[BOM+12] or connecting directly to the StreamAPI [Twi14c] we obtain a stream of tweets

2. **Data Pre-processing** - before going into the clustering algorithm, the data from each incoming tweet must be filtered and pre-processed in order to obtain the features we want the clustering to be based on

3. **Clustering** - we adapted the existing DenStream framework [CEQZ06] and added some features that allow, in a single pass, to summarize information about the tweets' text features as well as the numeric ones; this enables the algorithm to perform a multi-dimensional clustering over the tweets data, considering its content and spatio-temporal location.

4. **Visualization** finally, we developed a simple tool to visualize the results of our clustering; this tool allows for the visualization of information about each cluster as well as a world map visualization of where they are located.

This project is a natural "evolution" of the TweeProfiles tool, with its focus being on improving it so it could generate clustering results in real-time. Therefore, we can say that the main objective of this project was achieved. In the next chapter we will compare the differences betweent TweeProfiles 1 and 2 and discuss some of the decisions made.

5.2 TweeProfiles vs TweeProfiles2

In table 5.1 we can see a comparison of characteristics between TweeProfiles and TweeProfiles2.

Table 5.1: TweeProfiles vs TweeProfiles2 comparison

	TweeProfiles	**TweeProfiles2**
Dimensions	Spatial, Temporal, Social, Content	Spatial, Temporal, Content
Distance functions	Haversine, Time Difference, Geodesic, Cosine similarity	Haversine, Enclidean, Cosine similarity
Distance normalization	Min-Max	Min-Max
Distance combination	On-demand	Online step: mixed; Offline step: on-demand
Algorithm	DBSCAN	Micro-clustering: Hybrid DenStream; Macro-clustering: DBSCAN
Data Structures	Dissimilarity matrixes	Hybrid micro-clusters
Clustering process	Offline	Online & Offline

As was already stated, the differentiating characteristic of TweeProfiles2 is the ability to cluster of real-time. With this requirement come a lot of trade-offs and decisions that have to be made. We can say that TweeProfiles2 clustering process is leaner and faster (computing the dissimilarity matrices in TweeProfiles was a very costly and slow operation). However, that effiency comes with a price, as the clusters in TweeProfiles effectively represent a groups of tweets (you can know which tweets compose any given cluster), while the clusters in TweeProfiles2 represent a summary of the information of the tweets contained in the cluster (you cannot know which specific tweets compose a cluster). Also, we were forced to drop the social dimension, because it introduces the problem of how to normalize infinite distances. The solution proposed for this problem in TweeProfiles involves maintaining a social graph on runtime, which is impratical in a streaming environment, because of its size and we were now able to come up with a cost-effective strategy to do that.

5.3 Future work

Although we met the goal of our project, there still are some aspects of the work that could be improved, namely:

1. **Visualization** The visualization tool developed is rather rudimentary and could be improved in certain aspects, namely, its usability and ability to input test parameters.

2. **Text mining** The text mining techniques applied to the clustering process are rather basic (the tweets text pre-processing is done using *out-of-the-box* Lucene functions and the texts' representations are done using a simple *bag-of-words* model) and coul be improved.

For instance, we could try to apply a model of terms weighting similar to the one used in [BHP11]

3. **Clustering process** There are some parameters of the algorithm developed that were not thoroughly tested and could be fine-tuned if the time is spent doing the necessary testing. This would probablyproduce slightly better clustering results. Also, the distance combination percentages could be refined, because, even if all the distances are in a 0-1 scale, they don't have the same distribution; for instance, a difference of 0.1 in a cosine similarity function may mean the difference of a word, although the same 0.1 difference in the haversine function may mean a continent.

4. **Social Distance** As discussed above, the social dimension raises a problem that we were not able to find a solution for. It is an important issue that should be improved, because it could give us important insights on the social interactions that happen in Twitter.

References

[AHWY03] Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. A framework for clustering evolving data streams, 2003.

[Apa14] Apache. Apache lucene - https://lucene.apache.org/, 2014. [Online; accessed 1-July-2014].

[Bar02] D Barbará. Requirements for clustering data streams. *ACM SIGKDD Explorations Newsletter*, 3(2):23–27, 2002.

[BF10] Albert Bifet and Eibe Frank. Sentiment knowledge discovery in twitter streaming data. *Discovery Science*, 2010.

[BHKP10] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *J. Mach. Learn. Res.*, 11:1601–1604, 2010.

[BHP11] Albert Bifet, Geoffrey Holmes, and Bernhard Pfahringer. Moa-tweetreader: real-time analysis in twitter streaming data. *Discovery Science*, pages 46–60, 2011.

[Blo11] Twitter Blog. numbers - https://blog.twitter.com/2011/numbers, 2011. [Online; accessed 1-July-2014].

[BNG11] Hila Becker, M Naaman, and Luis Gravano. Beyond Trending Topics: Real-World Event Identification on Twitter. *ICWSM*, pages 438–441, 2011.

[BOM+12] Matko Boanjak, Eduardo Oliveira, José Martins, Eduarda Mendes Rodrigues, and Luís Sarmento. TwitterEcho: a distributed focused crawler to support open research with twitter data. In *Proceedings of the 21st international conference companion on World Wide Web - WWW '12 Companion*, page 1233, New York, New York, USA, 2012. ACM Press.

[CEQZ06] Feng Cao, Martin Ester, W Qian, and A Zhou. Density-Based Clustering over an Evolving Data Stream with Noise. *SDM*, pages 328–339, 2006.

[Cod14] Google Code. language-detection: Language detection library for java - http://code.google.com/p/language-detection/, 2014. [Online; accessed 1-July-2014].

[Cun13] Tiago Cunha. TweeProfiles : detection of spatio-temporal patterns on Twitter, 2013.

[EKSX96] Martin Ester, HP Kriegel, J Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *KDD*, 1996.

[Gam10] Joao Gama. *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC, 1st edition, 2010.

REFERENCES

[GMMO00] S Guha, N Mishra, R Motwani, and L O'Callaghan. Clustering data streams. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 359–366, 2000.

[Han76] J.C. Hannyngton. *Haversines Natural and Logarithmic Used in Computing Lunar Distances for the Nautical Almanac*. Great Britain. Nautical Almanac Office. G.E. Eyre and W. Spottiswoode, 1876.

[HKP11] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques, Third Edition*. Morgan Kauffman, 3rd edition, 2011.

[JSO14] JSON.org. Json - http://json.org/, 2014. [Online; accessed 1-July-2014].

[Kum12] A Anil Kumar. Text Data Pre-processing and Dimensionality Reduction Techniques for Document Clustering Sri Sivani College of Engineering Sri Sivani College of Engineering. 1(5):1–6, 2012.

[Lee12] Chung-Hong Lee. Mining spatio-temporal information on microblogging streams using a density-based online clustering method. *Expert Systems with Applications*, 39(10):9623–9641, 2012.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[MyS14] Dev MySQL. Connector/j - http://dev.mysql.com/downloads/connector/j/5.1.html, 2014. [Online; accessed 1-July-2014].

[SFB+13] Jonathan A. Silva, Elaine R. Faria, Rodrigo C. Barros, Eduardo R. Hruschka, André C. P. L. F. de Carvalho, and João Gama. Data stream clustering: A survey. *ACM Comput. Surv.*, 46(1):13:1–13:31, 2013.

[Twi14a] Twitter. About twitter - https://about.twitter.com/company, 2014. [Online; accessed 1-July-2014].

[Twi14b] Twitter. Rest api - https://dev.twitter.com/docs/api/1.1, 2014. [Online; accessed 1-July-2014].

[Twi14c] Twitter. The streaming apis - https://dev.twitter.com/docs/api/streaming, 2014. [Online; accessed 1-July-2014].