Geographical Impact of Microblogging Social Networks

Master Thesis

at

Department of Informatics
University of Fribourg

by
Roger Peter Kohler
Spiegelstrasse 4
3095 Spiegel
+41 79 813 42 73
ropeko@students.unibe.ch
Mat.-Nr. 08-114-969

supervised by
Prof. Dr. Philippe Cudré-Mauroux
and
Dr. Gianluca Demartini

February 9, 2014

Abstract

In this thesis the impact of microblogging social networks on the geographical distribution of bitly clicks is examined.

The messages of the social platform Twitter are combined with the geographical information, especially latitude and longitude, received by the clicks on bitly URLs. Bitly itself is a URL-shortener providing different statistics and is used intensively for microblogging services. The tweets are streamed during September 2013 and inserted with the bitly data provided by VeriSign into the used database system HBase. Then the geographical data is clustered with the software Weka. To reveal the impact of tweets on the bitly clicks, the clustering algorithm XMeans of Weka is applied on the bitly clicks before posting a tweet with a certain bitly URL and afterwards separately. The two clustering results are compared, especially the distortion of the clusters. The distortion difference is used to measure the influence of a tweet to the bitly clicks. Furthermore the distortion is directly related to the compactness in a cluster. The lower the distortion is the more local are the bitly clicks. Is the distortion very large, occur the clicks all over the world. In general the tweets make the bitly clicks more global.

Summarizing it was possible to show that the tweets impact the geographical distribution of the bitly clicks.

Acknowledgements

First of all I would like to thank my two supervisors Prof. Dr. Philippe Cudré-Mauroux and Dr. Gianluca Demartini, without them this work could not have been realized and for their interesting advices and helpfulness during my thesis.

During my master in computer science I could look in many exciting research fields. That is why I would like to thank all the professors and assistants too, which were able to keep and increase my fascination for computer science.

A special thank for Benoit Perroud and VeriSign. This work is done in partnership with VeriSign, which provided all the bitly data.

Further Alberto Tonon for setting up the four machines I used for this work and reinstalling Ubuntu on one machine after I removed an important system folder by mistake.

Finally to Jolanda Reusser for reading and correcting this documentation and for encouraging and motivating during difficult times, when the cluster was broken and all data needed to be inserted again.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Nowadays social media plays an important role in human life. Every hour there are millions of messages published by users and companies. This implies to an extremely high amount of data, called *Big Data*, and there are new technologies required for analyzing it in an efficient way. Furthermore the possibilities in studying and interpreting the data are nearly unconfined.

1.1 Motivation

There are a lot of studies already available, in which the messages of social platforms are considered and interpreted. Many of them examine objectives like popularity of events (Gupta et al. 2012) or sentiments in in the messages (Agarwal et al. 2011). But there are hardly any in the context of geographical information and distribution.

One of the most popular social networks currently is Twitter. The speciality of it is the microblogging aspect. All posted messages of Twitter have a limited number of characters. Each message, which is called *tweet*, can contain maximally 140 characters, so that the users need to adapt their language, which results in a high occurrence of abbreviations. Besides the written content of a tweet, Twitter stores a lot of additional information like the publisher, user mentions in the text, hashtags, language and many more. Although there is data about some geographic specific entities available, most of Twitter users do not use this properties.

For this work the company *VeriSign* provided data about bitly clicks. Bitly is a URL-shortener and records information about the clicks on a bitly URL. Especially in the context of microblogging social networks like Twitter, bitly is held in high esteem. For most registered clicks on a bitly URL, geographical information is present, notably for this work the parameters latitude and longitude.

If Twitter and bitly are combined by looking at tweets containing a bitly URL, both data is present.

1.2 Goals

One goal of this work is to analyze how much geographical information is existent in the tweets of Twitter containing a bitly URL. The main goal is to show if and how a tweet impacts the geographical distribution of the clicks of a bitly URL. Furthermore the degree of locality should be analyzed.

Besides the main goals concerning the geographical subject, the tweets and the bitly data should be compared and interpreted.

1.3 Organization

This thesis is split into three main parts. First the fundamentals (chapter 2) of the used components (Twitter, bitly, HBase and Weka) are explained. Next, the necessary steps of the implementation are shown (chapter 3). In that chapter first the concept of streaming tweets is discussed, followed by the data insertion into the used database system HBase and finally the clustering of geographical data is demonstrated. The last and most important part of this work consists of the results, where some general statistics about the whole amount of data and finally the geographical impact of a tweet to the bitly clicks are displayed and interpreted (chapter 5).

Chapter 2

Fundamentals

2.1 Twitter

In the last years social media increased their popularity extremely rapid. Today they are essential for keeping the social interaction and affect humans daily life. Some of the most famous social media are Twitter and Facebook. In contrast to Facebook, Twitter is a text-based microblogging application.

2.1.1 Tweets

A post on Twitter is called "tweet" and its character size is limited to 140. This restriction implies a modified language with many acronyms (e.g., "b4" for "before" or "ur" for "your"). Furthermore following conventions exist (Finin et al. 2010):

- Hashtags: Consist of a # token at the beginning (e.g., #switzerland, #cheese) of a word and underline important objectives in a tweet.
- User mentions: @ token followed by the name of the user (e.g., @rogerfederer). If the user mention is in front of the message it is a direct message at that user.
- Retweet: Symbolized by "RT" before the original tweet (e.g., RT original tweet text).

2.1.2 Additional Twitter based expressions

Twitter is a social network. A user has its own profile and can follow or be followed by another user. If user A follows user B, then user A is a follower of user B (the opposite is a following). Are two users followers and followings of each other they are called to be

friends. E.g.: Roger Federer posted in total 381 tweets, has 67 followings and 1'150'414 followers[1].

2.1.3 URL-shortening of Twitter

Another interesting feature is the URL-shortener. Often the number of characters is rather high for a URL[2] and with the character limitation of Twitter it is hard to post a URL. For this purpose a URL-shortener like bitly can be used[3]. Here is to mention that Twitter integrated its own shortener. The so called t.co URL wrapper transforms the URLs to a shorter version of up to 20 characters (inclusive http://t.co) (Twitter 2013). One difference between the t.co and bitly shortener is, that bitly returns a lot of additional statistical information like number of clicks, where and when the URL was clicked, from which device etc.. For this thesis especially the geographical positions of the bitly clicks are used.

2.1.4 Relevance of Twitter displayed in values

More than 200 millions tweets are published 2012 per day (Gao et al. 2012). In 2013 it already raised to more than 500 millions. This results in an average of about 5700 tweets per second. One interesting record was achieved at the beginning of August 2013: 143'199 tweets per second during watching a movie in Japan (Twitter Engineering Blog 2013). But meanwhile there is a high probability that this record is already scaled up and will raise further on.

2.1.5 Censorship of different countries of Twitter

Another fact to mention is the censorship in different countries. Because of political reasons in some countries Twitter was or is inaccessible. Today it is blocked in China, where Sina Weibo is the leader of the offered microblogging services (Gao et al. 2012).

[1] Read out on December 16th 2013
[2] See list of abbreviations, page 62
[3] More information about bitly in the next chapter

2.2 Bitly

Bitly is one of the most used URL shortener. Each month more than 4 billion times a bitly link is clicked. In addition the bitly Inc. offers different statistics for each link, like number of clicks over time, country, the social media platform a link was clicked etc.. This helps web publishers to analyze and increase their social media traffic (Bitly 2014).

To show the procedure of the shortener it is assumed, that the URL to shorten is like http://www.website.com/this/is/a/long/url/without/using/any/shortener. Well if a user would like to publish this URL in a microblogging platform like Twitter too many characters are necessary and the user can not write a lot of comments besides the long URL in the tweet. Therefore, the user utilizes a URL-shortener and instead of the above URL a short version like http://bit.ly/XyZ34a is created. If the new URL is entered in a browser it is automatically redirected to the original content. Just a "+" is added to the end to show the analytics of a bitly link (e.g. http://bit.ly/XyZ34a+).

Furthermore, there is a paid possibility to use the own domain. Especially enterprises build their own domains like pep.si (Pepsi) or nyti.ms (New York Times) instead of bit.ly (Wikipedia 2014). The benefits for the companies are to obtain a personal touch, it is an easier way to recognize the creator and of course advertisement.

Thanks to VeriSign a lot of information for each click on a bitly link is available. The data is stored in semi-structured JSON format, which is easy to read and write for humans and easy to parse for machines (JSON 2014). In listing 2.1 an example of the data a click on a bitly link generates can be found.

There are information about the used device ("a"), geographical position (country code: "c", time zone: "tz", geographical region: "gr", city: "cy" and latitude/longitude: "ll"), timestamp ("t", "hc"), identification ("i"), accepted languages ("al"), URL to the original ("u"), bitly hash ("g", "h") etc. (for the user's privacy some values of the example were changed). The analytics to this bitly link can be found as described above on http://bit.ly/Kwidys+.

```
{
        "a": "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.5 (
            KHTML, like Gecko) Chrome/19.0.1084.52 Safari/536.5",
        "c": "BR",
        "nk": 0,
        "tz": "America/Sao_Paulo",
        "gr": "27",
        "g": "Kwidys",
        "h": "Kwidyr",
        "k": "4fc83f97-002ba-07574-3d1cf10a",
        "l": "tweetdeckapi",
        "al": "pt-BR,pt;q=0.8,en-US;q=0.6,en;q=0.4",
        "hh": "bit.ly",
        "r": "http://www.facebook.com/l.php?u=http%3A%2F%2Fbit.ly%2
            FKwidyr&h=qAQFxenRiAQGvkzhiC4Ux0mNsa_eJw-Z1yT5CyROSdzp3oh",
        "u": "http://www.moluscontos.com/2012/06/velocidade-da-luz/",
        "t": 1338523543,
        "hc": 1338523356,
        "cy": "SPaulo",
        "ll": [ -24.533300, -45.616699 ],
        "i": "3a2fa6e358b6f11e29afd30705656c5f"
}
```

Listing 2.1: JSON-formatted information generated by a click on a bitly link

2.3 HBase

Some years ago, the main database systems were the relational database management systems (RDBMS). With the increasing speed of the Internet, the social media got more and more popular. The new reached importance of them opened a lot of questions. What is the most efficient way to store all the new generated data? How can the big amount of data be analyzed without consuming too much time? How must thousands of storage machines be connected for a fast interaction between them or to access on its data rapidly? The RDBMS got to their boundaries with the big data and new concepts are needed. HBase is one of them and its architecture relies on different parts like HDFS and Zookeeper which are introduced in the next sections.

2.3.1 Hadoop

In (White 2012, p. 3) the problem and the reason to make alive the project called Hadoop is explained as follows:

> *The problem is simple: While the storage capacities of hard drives have increased massively over the years, access speeds – the rate at which data can be read from drives – have not kept up.* (White 2012, p. 3)

Since January 2008 Hadoop belongs to the projects of the Apache Software Foundation, which supports open source software projects. A world record was broken in April 2008, by sorting one terabyte of data as fast as possible. Hadoop managed it in 209 seconds which was more than one minute faster than the winner of the previous year. The execution happened on a 910-node cluster. (White 2012, pp. 11–12).

The Hadoop project is split in many modules. HBase is one of them. Different well-known companies like Facebook, Twitter, Yahoo!, Adobe etc. use HBase (George 2011). But before looking on the distributed database system HBase, a short introduction of the main components HDFS and Zookeeper should be done, which are subprojects of Hadoop too.

2.3.2 HDFS

In some publications Google described an improved storage and processing system. The interesting part of this system is, that it can be achieved with the wide-spread hardware and does not need any optimized and therefore expensive machines. Furthermore, the system is scalable. These ideas were implemented by the open source Hadoop project

and are called HDFS and MapReduce (George 2011, pp. 1–2). HDFS stands for Hadoop Distributed File System. The advantage of HDFS is, that it can run on large clusters of commodity machines to store very large files. It is optimized rather for reading than writing data. Therefore, a dataset is copied from the source (write-once) and then analyzed over time (read-many-times). (White 2012, pp. 45–46)

2.3.3 Zookeeper

Another important component of HBase is Zookeeper. While HDFS is responsible for the distributed storage of large data on numerous machines, Zookeeper's working area is the distributed coordination. It ensures that only one master is running and stores the bootstrap location, so that the master is able to discover the region servers. The goal is to keep the reliability and availability of all machines in a cluster as high as possible. (George 2011, pp. 25–26)

2.3.4 Architecture of HBase

HBase is build on HDFS and Zookeeper. Each node in a cluster of machines has to interact with a Zookeeper server installed on at least one machine. Although there is a possibility to use the normal file system, it is more for experimenting at the beginning just with one machine. As soon as there are more nodes, it is recommended to use HDFS. That is because every node should need HDFS as file system. Furthermore one machine has to act as the master of the cluster. This node organizes and recovers failures of the region servers and assigns the regions declared in the file *conf/regionservers* to the registered nodes. Other important files during setting up a cluster are *conf/hbase-site.xml* and *conf/hbase-env.sh*, where most of configurations can be changed. For this thesis the Cloudera Manager is used, which simplifies the setup of an HBase cluster (see section 2.3.7). (White 2012, pp. 459–461)

In figure 2.1 an example of an HBase cluster with its components is displayed. For this thesis three region servers and one master are used, where one of the region servers runs on the same machine like the master.

2.3.5 Conceptual view of an HBase table

An HBase table consists of row keys, column families and timestamps. The row key is the identifier and has to be unique. It is used for finding quickly the entries on the region servers. Each column family can have millions of columns. The number of column families

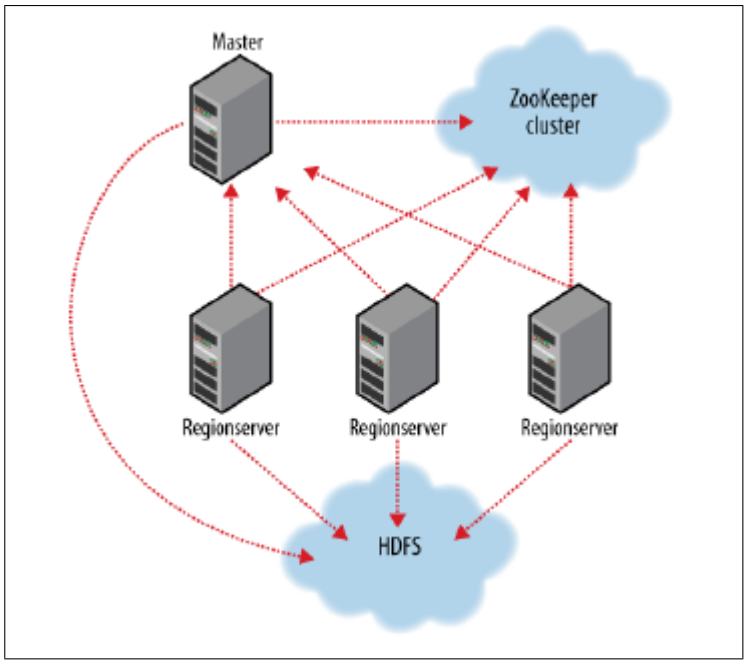

Figure 2.1: HBase cluster with three nodes operating as region servers and one node representing the master. Each of them uses the services Zookeeper and HDFS (source: White 2012, p. 460).

should be kept small, because it is possible to do compactions of the tables. This is done per region server. If one column family holds most of data, while another one is nearly empty, the same amount of compaction work is done for the small family too. Therefore, it is the best way just to have one column family, which can contain as many columns as wanted. (HBase 2014, section 6.2)

In table 2.1 a conceptual view of an HBase table is shown. Each entry needs at least a row key. If somebody puts a value into the table, it inserts an entry of the form *columnFamilyName:columnName* with a value. While the column family has to exist already, the specified column can be a new one. Additional for each inserted value a timestamp is stored. Therefore, different versions of a column value are possible for an equal row key (in table 2.1 the row key *"id123"* has two values for the column *city* in the *geo* column family, but with different timestamps). The maximum number of versions can be specified; by default it is set to 3, which allows three versions. If a fourth value is added to the same column, the one with the oldest generated timestamp is removed.

2.3.6 Physical view of an HBase table

Instead of storing row by row like in the traditional databases (i.e. in RDBMS), column-oriented databases won influence. The column-based approach was implemented by HBase

Row key	**Timestamp**	**Column family** user	**Column family** geo
"id123"	1389609428492	user:name="Roger"	geo:city="Berne"
"id123"	1389609428492	user:height="179"	
"id345"	1389609544052	user:name="John"	
"id123"	1389609429518		geo:country="Switzerland" geo:city="Spiegel"

Table 2.1: Conceptual view of an example of an HBase table (edited source: HBase 2014, section 5.1).

too, but not in the usual way. Not the table itself is column-oriented. The main aspect of HBase is, that it stores the data on the disk in a column-oriented format. (George 2011, p. 3)

The physical view of an HBase table is rather different than the conceptual. In many other database systems, empty cells are marked in the file system to be empty (i.e. with the value *null*). In HBase empty cells are just ignored and not stored to the file system. With this concept a lot of storage can be saved. The tables are physically stored on column family basis, which is shown in the tables 2.2 (column family *user*) and 2.3 (column family *geo*).

Row key	**Timestamp**	**Column family** user
"id123"	1389609428492	user:name="Roger"
"id123"	1389609428492	user:height="179"
"id345"	1389609544052	user:name="John"

Table 2.2: Physical view of the example HBase table 2.1 based on the column family *user* (edited source: HBase 2014, section 5.2).

Row key	**Timestamp**	**Column family** geo
"id123"	1389609428492	geo:city="Berne"
"id123"	1389609429518	geo:country="Switzerland" get:city="Spiegel"

Table 2.3: Physical view of the example HBase table 2.1 based on the column family *geo* (edited source: HBase 2014, section 5.2).

2.3.7 Cloudera

Cloudera provides the Apache Hadoop-based software with its components and offers a data platform (Cloudera 2014). To simplify the life in configuring HBase, Hadoop and Zookeeper manually, the Cloudera manager was developed. It supports the setup of a cluster with many nodes too and as soon as the cluster is running, for each instance in the cluster some statistics are available. Is i.e. a region server shut down, then a warning occurs in the manager. In figure 2.2 a screenshot of the interface is shown. The manager itself is only installed on the master machine with some management services and coordinates all the configurations to the region servers. For this thesis only a small amount of Hadoop's components are installed. Zookeeper and HDFS are required for using HBase. Therefore the installed services are: HBase, Zookeeper, HDFS and of course the management services.

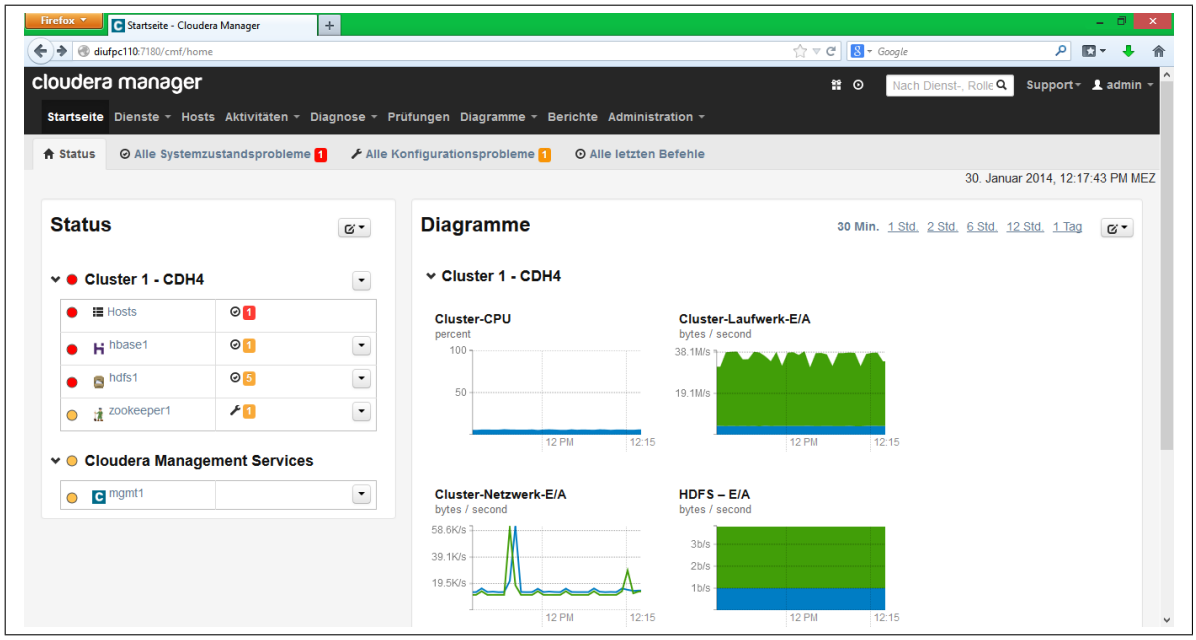

Figure 2.2: Cloudera manager interface

2.3.8 HBase shell

Command	Explanation
hbase shell	starts the shell of HBase
create 'data', 'user'	creates a new table called *data* with one column family *user*
list	lists all existing tables (in this example the table *data*)
put 'data', 'id123', 'user:name', 'John'	inserts a new column *name* with the value *John* to the column family *user* and the row key *id123* (because the row key does not exist already, it will create a new entry)
get 'data', 'id123'	looks for the row key *id123* in the table *data* and returns all columns of this row (in this example the column *name* with the value *John*)
scan 'data'	returns all rows with its columns
disable 'data'	disables the table, so that no *put*, *scan* or *get* can be done
drop 'data'	removes the table (only possible if it was disabled before)
exit	finishes the HBase shell

Table 2.4: Most important commands of HBase shell (create table, put and get a value, remove table). (HBase 2014, section 1.2)

As soon as HBase runs on the machines of a cluster, it is possible to try out the HBase shell. To start it, the command *hbase shell* has to be entered in a terminal. The shell is especially of advantage if the user only wants to do some easy queries like read a row or put a value into an HBase table. The most important commands are *get*, *scan* and *put*. In table 2.4 the process of creating a table, put and read values into the table and finally remove a table are shown.

2.3.9 HBase using the Java API

The commands of the HBase shell (section 2.3.8) can be implemented with Java too. Listing 2.2 executes the same commands like in the shell (see table 2.4). Lines beginning with a // are comments and explain the Java programming code of the next line.

```
20 // Creates a configuration with the HBase resources
21 Configuration configuration = HBaseConfiguration.create();
22 // For managing HBase tables
23 HBaseAdmin admin = new HBaseAdmin(configuration);
24
25 // Holds all details like column families
26 HTableDescriptor table = new HTableDescriptor("data");
27 // Adds a new column family to the table descriptor
28 table.addFamily(new HColumnDescriptor("user"));
29 // Creates the HBase table definitively
30 admin.createTable(table);
31
32 // Lists all the table names
33 admin.listTableNames();
34
35 // Used to communicate with the specified table
36 HTable table = new HTable(configuration, "data");
37
38 // Creates a Put object with the row key "id123"
39 Put put = new Put(Bytes.toBytes("id123"));
40 // Adds the value "John" for the column "name"
41 put.add(Bytes.toBytes("user"), Bytes.toBytes("name"), Bytes.toBytes("
       John"));
42 // Finally applies the Put to the table
43 table.put(put);
44
45 // Creates a Get object for the row key "id123"
46 Get get = new Get(Bytes.toBytes("id123"));
47 // Gets the row with its column from the table
48 Result result = table.get(get);
49
50 // Creates a Scan object
51 Scan scan = new Scan();
52 // Scans through the table "data"
53 table.getScanner(scan);
54
55 // Disables the table
56 admin.disableTable("data");
57 // Removes the table
58 admin.deleteTable("data");
59
60 // Closes the admin session
61 admin.close();
```

```
62 // Closes the table session
63 table.close();
```

Listing 2.2: Java implementation of the same commands as in the table 2.4

Especially for a scan of a table, there exists a lot of quite helpful options. Without setting any properties the scan is executed to the whole HBase table. If the table itself is millions of rows large, it consumes a lot of time. For this problem the options *setStartRow* and *setStopRow* exist. Because of the column-oriented storage architecture for HBase tables, it is fastest to specify the row key. With these options the scan walks from the specified start to the end row what is much more efficient than scanning the whole table. Another interesting option is *setCaching*. By default the cache value is set to 1. For every row it will call back to the (region) server (HBase 2014, section 12.9.1). But if the goal is to count the number of rows of a table, it is advantageous to raise this value to about 10000. So it caches 10000 rows and is much faster in counting. Furthermore, a scan is able to have filters. If the scan should only return all the rows with a column value smaller than a factor, the implementation is like in listing 2.3. In this example the resulting rows are all with a smaller value than 10 for the column *columnName*. Of course a scan can have several filters with the changeable property of *MUST_PASS_ALL* or *MUST_PASS_ONE* filter.

```
1 SingleColumnValueFilter filter = new SingleColumnValueFilter(Bytes.
    toBytes("columnFamilyName"), Bytes.toBytes("columnName"), CompareOp.
    LESS, Bytes.toBytes("10"));
2 scan.setFilter(filter);
```

Listing 2.3: Filter example of an HBase table scan

2.4 Weka

The software used for the analysis of the tweets and bitly data is called Weka and stands for *Waikato Environment for Knowledge Analysis*. It was developed by the machine learning group at the university of Waikato. There are a lot of machine learning algorithms available, which can be applied to a dataset. (Weka 2014)

2.4.1 Weka's explorer

At the beginning of working with Weka it is recommended to use the provided explorer (figure 2.3). Several measurements are computed automatically like mean or standard deviation. With a small data set some of the algorithms can be executed and with the visualization tool a very helpful overview of the data is given. For each algorithm there exist many parameters the user is able to change. Trying out the functionalities of the explorer is the easiest way to estimate the possibilities of Weka.

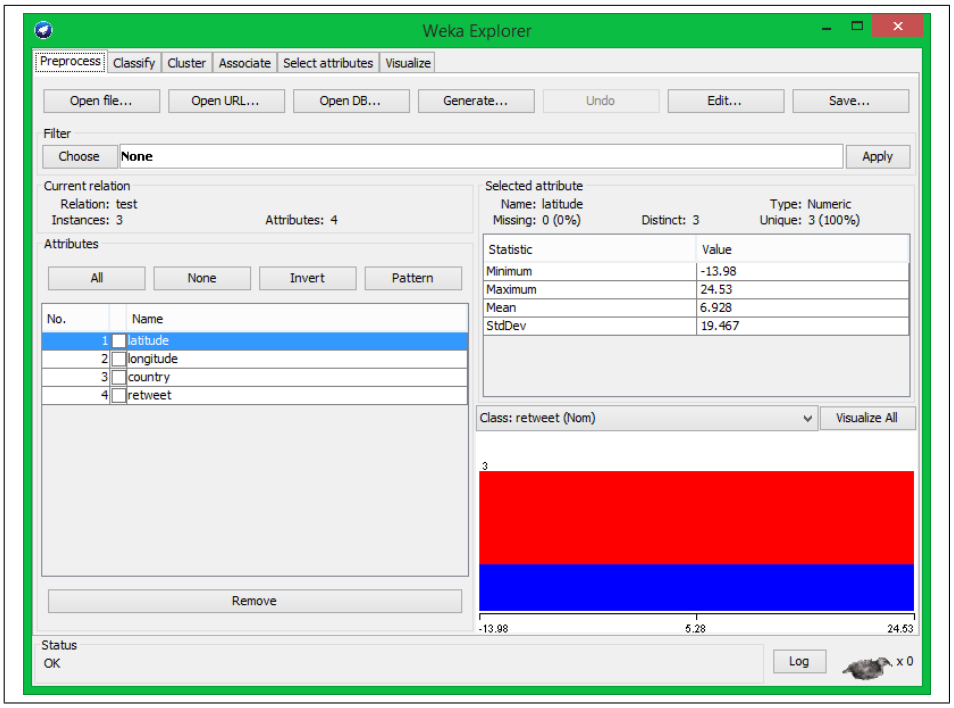

Figure 2.3: Weka explorer with the loaded .arff example file of listing 2.4

Besides the given explorer, all the algorithms can be implemented with Java. In this thesis the Java library of Weka is used especially for clustering the geographical information of the data.

2.4.2 ARFF-File

Weka uses .arff-files for their data sets, which stands for *Attribute-Relation File Format* (Weka 2014). Nevertheless the .csv (comma-separated values) file extension is supported too and can be loaded into the explorer. It is an advantage to load directly .arff-files, because of the information level. In Weka each instance of a data set has one value to a defined attribute. If a .cvs-file is used, this attribute information is missing. In an .arff-file each attribute is defined at the beginning and can have several types like numeric, nominal, string, etc.. For a small data set, where the user knows which value belongs to which attribute, the .cvs-file does not make any problems, because the values are translated automatically into a data type. But as soon as the data set is larger and it is hard to keep the properties of all values in mind, it is easier to define the attributes directly in the file. So the user does not have to care anymore for the data type of all the values.

An .arff-file is separated in three parts: First the name of the relation is written (@relation nameOfRelation). Afterwards the attributes are declared (@attribute nameOfAttribute dataType) and finally the data itself follows (@data). (Weka 2014)

In listing 2.4 an example of an .arff-file is shown.

```
1  @relation test
2
3  @attribute latitude numeric
4  @attribute longitude numeric
5  @attribute country string
6  @attribute retweet {true, false}
7
8  @data
9  10.234, 20, BR, true
10 24.53, 142, CH, false
11 -13.98, 21, USA, false
```

Listing 2.4: Example of an .arff-file used in Weka (loaded to the explorer of Weka in figure 2.3)

Chapter 3

Methods

In this chapter the implementation is presented. Figure 3.1 shows the different steps to reach the final goal of analyzing the clustered geographical distribution of the bitly data depending on the time a tweet was posted. In the first section, it is explained how the tweets of Twitter are streamed with the thematic priority on the authorization. Next, the programmatic implementation and problems of inserting the data into HBase are discussed. After the insertion the data is ready for clustering with Weka. The last step of analyzing the clusters and its parameters is done in chapter 5.

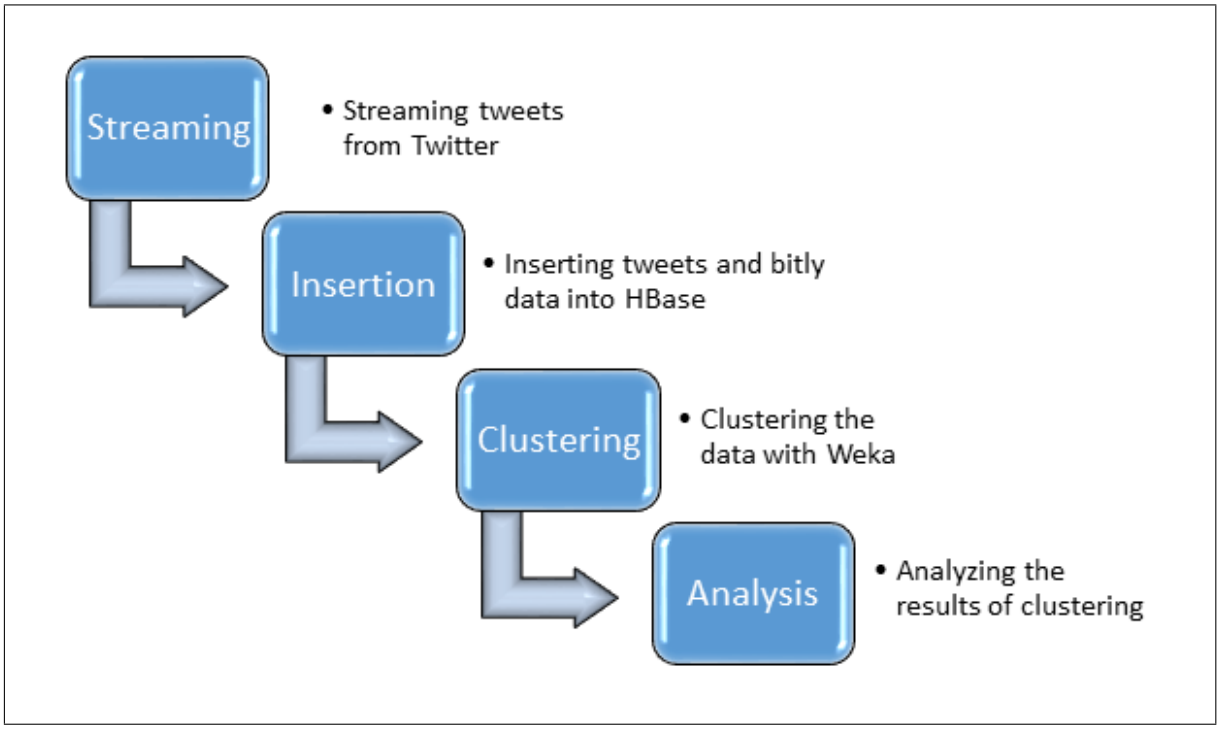

Figure 3.1: Processes that are needed to get the impact of a tweet on the geographical distribution of bitly URLs

3.1 Streaming tweets with the Twitter API

Before any analysis of the tweets can be started, the tweets are needed. To stream for tweets the Twitter API is used. The Java library *twitter4j* supports most operations specified in Twitter. But before explaining the method for streaming tweets, it is important to know exactly which tweets are wanted.

3.1.1 Constraints for data to stream

There are several thousands tweets published per second, most of them are not interesting for the analysis and will fill storage and consume processing time. For this reason first some constraints have to be declared to restrict the streamed tweets:

- Tweets: Solely tweets containing at least one bitly URL are streamed.
- Bitly: Only original domains with bit.ly or bitly.com are collected (i.e. company specific bitly links like pep.si/... are not taken into account).
- Retweets: Just one iteration is done for retweets. This means that only direct retweets of the original message are collected and retweets of another retweet are not considered.
- Time: The publication date has to be in September 2013.

Correct streamed tweet examples would be (assuming that all examples are published in September 2013):

This is an example of a correct tweet with a bitly URL http://t.co/ABC123xyz!
RT This is an example of a correct tweet with a bitly URL http://t.co/ABC123xyz!

The automatically changed URL by the t.co-shortener of Twitter refers to a bitly link like http://bit.ly/XyZ34a. If the t.co-URL is correlated to another generic domain (i.e. http://nyti.ms/abc56D) it would be get caught by the filter.

3.1.2 Authorization for streaming tweets

Since June 2013 the unauthorized streaming of tweets is not supported any longer, because the API version 1 is replaced by 1.1. In this new version a lot of restrictions are introduced and for many functions the user needs an authorization. That is why the programmed code does not work any more and an adaption was necessary. Fortunately, the Twitter

API version 1.1 was already established some time ago and the Java library *twitter4j* was customized for the new version.

There was a user registration required and in addition for doing operations like streaming tweets an application had to be registered too. Because it is always possible that an electronic machine shuts done without any prior warning, a second application to stream during the same time on a different machine was registered. This preventively made redundancy was finally more than needed. During streaming tweets, on one machine the storage was full and it could not store more tweets.

After the registration of a profile and of the applications, there is one last step to do. Each application needs an access token. To request such a token the program sends the consumer key and password to Twitter (these two values were given after registration of the application) and an access token with a password is generated. The programming code for this purpose can be found in listing 3.1. First a request token is received. Then in line 52 an URL is generated automatically. After opening the URL in a browser, a PIN is displayed. Back in the program this PIN has to be entered and if the PIN is correct the access token is built.

```
50 RequestToken requestToken = twitter.getOAuthRequestToken();
51 BufferedReader reader = new BufferedReader(new InputStreamReader(System.
       in));
52 System.out.println("Open the following URL and grant access to your
       account:" + requestToken.getAuthorizationURL());
53 System.out.print("Enter the PIN:");
54 String pin = reader.readLine();
55 AccessToken accessToken = twitter.getOAuthAccessToken(requestToken, pin)
       ;
```

Listing 3.1: Request of an access token for authorization of several functionalities of Twitter API 1.1.

3.1.3 Java programming code to stream tweets

Finally after receiving the access token it is possible to execute authorized operations like streaming. In listing 3.2 the programming code is shown for streaming tweets. First a twitter stream object is instantiated. In line 103 the authentication is now possible, after receiving the access token like discussed in the previous paragraph. Another instantiation is done in line 107. A filter object is initialized, which is essential to limit the tweets that

are streamed. This filter uses the keywords "bitly" and "bit ly" to stream. The second keyword is necessary to get all tweets with URLs containing the domain bit.ly. If the second keyword is "bit.ly", which would be more logical, it would not stream tweets with a bitly link for any reason. This implies that if in a tweet the words "bit" and "ly" occur, the tweet is streamed too.

```
100 public Stream(OAuth o) {
101         TwitterStream twitterStream = new TwitterStreamFactory().
                getInstance();
102         o.setTwitter(twitterStream);
103         o.authenticate();
104         ArrayList<String> tweets = new ArrayList<String>();
105         StatusListener listener = new BitlyListener();
106         twitterStream.addListener(listener);
107         FilterQuery query = new FilterQuery();
108         // bit ly: bit.ly, bitly: bitly.com
109         String[] keywordsArray = { "bit ly", "bitly" };
110         query.track(keywordsArray);
111         twitterStream.filter(query);
112 }
```

Listing 3.2: Streaming tweets with Twitter API 1.1.

To avoid that the mentioned tweets with the words "bit" and "ly" are streamed, a status listener is defined. For each tweet which overcomes the filter, because it contains one of the two specified keywords, it will be further tested. First the method *containsBitlyURL()* (see listing 3.3, lines 82-89) looks inside the URL entities of the tweet if there exists at least one correct bitly link of the format "http://bit.ly/*", "https://bit.ly/*", "http://bitly.com/*" or "https://bitly.com/*" (tested in the method *isBitlyURL()*, lines 92-93). The * stands for "followed by any sequence of characters". Does the method return true, then there exists one or more correct bitly links and it is added to the streamed tweets. If no bitly URL is inside of the tweet, a second possibility is available. Twitter does not state a bitly link of a retweet in its URL entities. Instead the original tweet, which was retweeted, is given as a status object and from this object the URL entities are accessible as well. Hence the same conditions can be applied to the original tweet (line 75). Summarizing, either the tweet contains directly a bitly link or it is a retweet, where the original tweet has a bitly link.

```
70 @Override
71 public void onStatus(Status status) {
72         boolean add = false;
73         add = containsBitlyURL(status.getURLEntities());
74         if (!add && status.getRetweetedStatus() != null) {
75                 add = containsBitlyURL(status.getRetweetedStatus().
                       getURLEntities());
76         }
77         if (add) {
78                 addTweet(status);
79         }
80 }
81
82 private boolean containsBitlyURL(URLEntity[] urls) {
83         for (URLEntity url : urls) {
84                 if (isBitlyURL(url.getExpandedURL())) {
85                         return true;
86                 }
87         }
88         return false;
89 }
90
91 private boolean isBitlyURL(String url) {
92         return url.contains("http://bit.ly/") || url.contains("https://
              bit.ly/") || url.contains("http://bitly.com/") || url.
              contains("https://bitly.com/");
93 }
```

Listing 3.3: Validation of the URLs to be a bitly link or not.

If a comparison with the constraints is made at this point, all except the time-specific are already fulfilled. The condition that the tweets are from September 2013 is accomplished, because the streaming happens in real-time and as soon as a new status is posted it will be tested. Accordingly for the time constraint the streaming algorithm needs only to be executed during September.

3.1.4 Storing streamed tweets to files without out-of-memory exception

One aspect that has to be considered during storing the data to text files is the out-of-memory exception. Every second there are more than forty tweets posted, which contain a correct bitly URL. If this amount of data is kept over time, the usage of memory increases and finally the execution is broken because of an out-of-memory exception. For this purpose the streamed tweets have to be stored frequently. Testing the boundaries by counting the tweets and waiting for the out-of-memory exception returned in a limit of about 150'000 tweets. To avoid the exception and the consequent termination of the execution, the maximum number of tweets in an array is set to 100'000. As soon as this value is reached, the streamed tweets are stored to a text file and the array is cleared. The 100'000 tweets are achieved after about 35 minutes. If this is projected, the number of tweets per day gets a total of about 4.2 millions.

Another idea was to store the tweets every 30 minutes. This approach would simplify the inventory of the files. But there is one negative point in this attempt: Some tweets arrive with a small delay of i.e. one second. If such a tweet is exactly at the time the file is created, it can happen, that the tweet is stored on the next file. Consequently the file does not represent only the tweets of the 30 minutes. Instead it can contain other tweets with a delay. This is the reason why the idea was discarded. A different positive fact of the first approach is that there is a one-hundred percent certainty that exactly 100'000 tweets are stored in the file and it is easy to count the total number of streamed tweets.

3.2 Insertion of tweets and bitly data into HBase

Before the insertion of the data into HBase can be started, there are some thoughts on the dependencies needed, which implies to the order of the inclusion to HBase. Furthermore, the streamed tweets are stored in a special format, which is not readable by the used JSON java library. Therefore a conversion of the tweets is inevitable. Those steps are shown in figure 3.2 and discussed in the next sections.

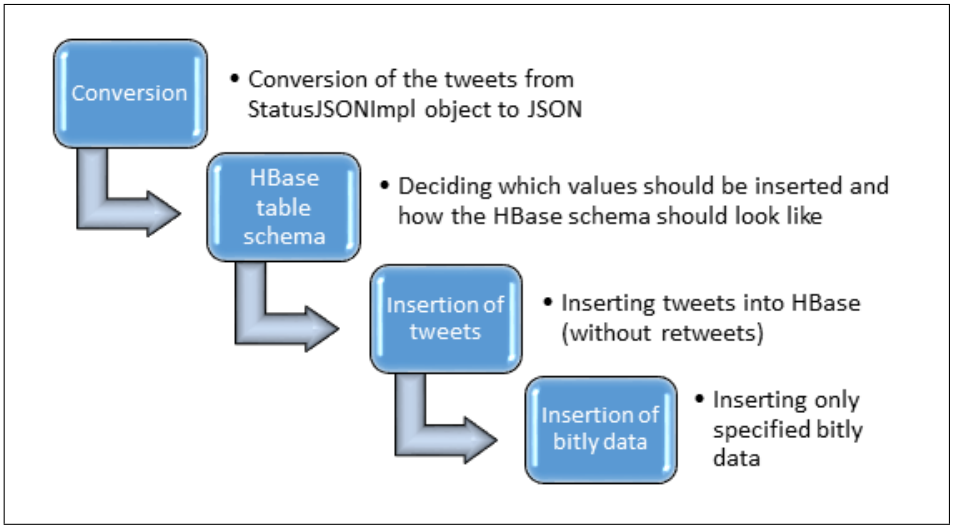

Figure 3.2: Required steps to insert all needed data into the HBase tables

3.2.1 Dependency of the tweets and bitly data

To decide if the order of inserting the data into HBase plays a role, the dependencies of the components have to be considered. First the bitly data is examined, because not all the data should be moved into the HBase table. The streamed tweets contain bitly URLs and only the data for this URLs is needed. Consequently, it is necessary to put the tweets to HBase before the bitly data. As soon as all the streamed tweets are included to HBase, it is possible to verify if a bitly URL occurs in at least one of the tweets.

3.2.2 Conversion of the streamed tweets

The streamed tweets are stored as *StatusJSONImpl* objects in text files. Each line belongs to one tweet. If the file is read line by line with Java, a tweet is available as String object. Unfortunately the *twitter4j* library does not support initializing of the *StatusJSONImpl*. Therefore, the tweet has to be converted into the general JSON-format. In listing 3.4 the conversion is displayed.

```
1  // The streamed tweet as StatusJSONImpl object
2  StatusJSONImpl{
3          createdAt=Sun Sep 15 14:36:47 CEST 2013,
4          id=369075402326540290,
5          text='Text of the tweet with the bitly URL http://t.co/
               JbH5S0Rb81',
6          ...
7  }
8
9  // Converted tweet to the general JSON format
10 {
11         "createdAt":"Sun Sep 15 14:36:47 CEST 2013",
12         "id":"369075402326540289",
13         "text":"Text of the tweet with the bitly URL http://t.co/
               JbH5S0Rb80",
14         ...
15 }
```

Listing 3.4: Conversion of a streamed tweet from *StatusJSONImpl* to the general JSON format

3.2.3 HBase table schema

Next and last step before the insertion can be started is the creation of the HBase table. It is relevant to know which information about the tweets is needed, because it makes no sense to store everything. There are a lot of useless information like *profileBackgroundColor* of the user (at least for this thesis). Furthermore the information is separable into several subjects. On the one hand there exists information about the tweet itself like time of publishing, text of tweet, hashtags, etc.. On the other hand there are user or place specific information like followers, friends, statuses (tweets) of user or country code for the place. This fact automatically implies to the HBase schema. Three tables can be created: Tweet, user and place. Because there are millions of tweets and the storage is limited, it is important to keep the names of the tables and columns as short as possible. The tweet table will be renamed to t, user to u and place to p. The column family is in each case f and unique per table. The explanations of all the abbreviated column names are shown in table 3.1.

The separate tables need at least one column which exists in another table. I.e. the tweet table contains the columns *uid* and *pid*. These are the row keys for the tables user

Column	Explanation
tweet table *t*	
tid	identification of the tweet (row key)
ts	timestamp, when the tweet was published
te	text of the tweet
lat	latitude (not available for most of the tweets)
lon	longitude
ht	hashtags
um	user mentions in the tweet
uid	user identifier
pid	place identifier
bid	bitly URL
user table *u*	
uid	user identifier (row key)
frc	number of friends (friends count)
foc	followers count
fac	favorites count
lc	lists count
sc	statuses count (published tweets)
la	language
lo	location
tz	time zone
utc	coordinated universal time
place table *p*	
pid	place identifier (row key)
cc	country code
ci	name of country (country identifier)
na	name of place
ty	type of place (i.e. city)

Table 3.1: Abbreviations of all column names, sorted by HBase tables

and place. It is important that the column that links two tables together is in one of them the row key. Otherwise if the tweet with a *pid* is given, it is a must to scan the whole table for the value. But if *pid* is the row key, then the information is found in a heartbeat. Now the question appears, what would happen if the query is reversed. The *pid* is given and it is asked to find all tweets with this place value. With the three tables created, the query has to scan all the tweets and takes plenty of time. The problem can be solved by creating a help table. This table needs again *pid* as row key, but the columns are different to the place table. The column names are now the tweet identifiers (*tid*). If the above query is tested again it looks in this helping table and gets immediately all the identifiers of the tweets, which have *pid*. The same is done for the user table. The row key is *uid* and the column names are the *tid*s.

At this moment there are six tables planned. However, before the insertion can be started, it is better to look forward to the tables needed for the bitly data. Unfortunately, the identifier is not *bid*, because these are only the shortened URLs of the bitly link (without http://bit.ly/) and each URL can be clicked many times. Therefore, the identifier and row key for the bitly table is the identifier of the click (abbreviated with *brk*, see table 3.2). With the thoughts from the previous paragraph it seems clearly to create a help table too. This table has the row key *bid* and the column names *brk*. Henceforth the tables of the bitly and tweet data are connected. The only missing case is if the *bid* is given and the query is to find all tweets with a *bid*. For this just another help table is necessary with *bid* as row key and *tid* as column names. Now the total number of tables is raised to 8.

Column	**Explanation**
bitly table *b*	
brk	bitly identifier for each click (row key)
bts	timestamp, when the bitly URL was clicked
blat	latitude of bitly click
blon	longitude
bcc	country code
ci	city
gr	geo region
btz	time zone
al	accepted languages

Table 3.2: Abbreviations of all column names of the bitly table *b*

To keep the overview of all the HBase tables and their connections, figure 3.3 is displayed.

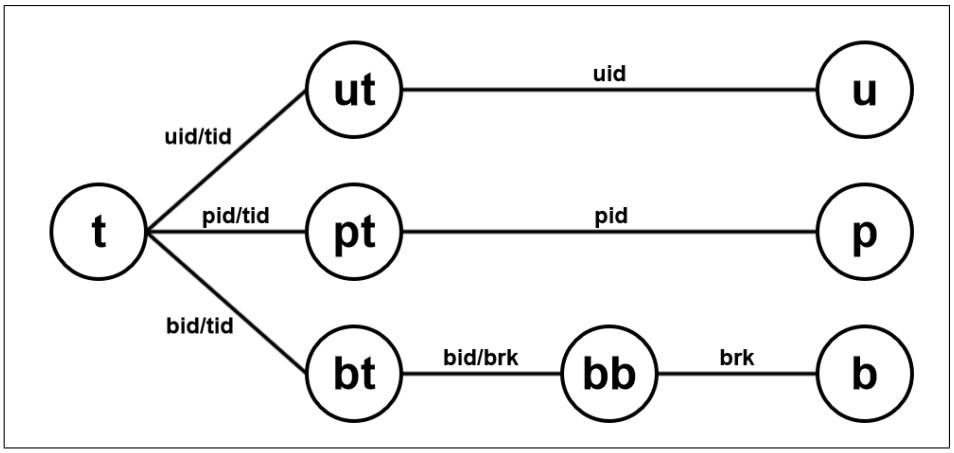

Figure 3.3: HBase tables with their connectivities

At this point an example of a query should be done to explain the figure 3.3 and the schema of the HBase tables. The goal is to find the number of followers of a user (*foc* of table u). Given is only the identifier of a bitly URL (*bid*). The start is in table bt, where the row keys are the *bid*s. As described earlier in this chapter, the column names of table bt are the *tid*s. Hence it is possible to look in t for the tweet with the row key *tid*. In its information the column for the user identification exists (*uid*). With the *uid* it is feasible to get the user information directly of table u and the number of followers as well. The whole query can be done only with Gets. None of the table need to be scanned. This shows that it is important to know what the row key of each table is.

The HBase tables exist at this point only theoretically. Therefore it is time to create them in HBase. There are two possibilities, either using the HBase shell or the Java API of HBase. In the shell itself it is rather easy. The command for creating the table is just "create 'tableName', 'f'", where *tableName* has to be replaced with t, u, ut, etc.. But because all other tasks are done with Java, the creation should happen with it too. With the programming code of listing 3.5, it creates a table called *tableName*, which is the only parameter to change to the real table names.

Last thing to discuss before the insertion can be started is the version number. By default it is set to 3. Looking on the information and the HBase tables, it gets clear that this value is unnecessarily high. For all tables (without the tweet table t) it makes no sense to allow several versions. If the value of different versions is set to 1, it replaces the existing information. In other words if a column of a row is preexisting, then the value for it is just overwritten.

```
1 Configuration configuration = HBaseConfiguration.create();
2 HBaseAdmin admin = new HBaseAdmin(configuration);
3 HTableDescriptor table = new HTableDescriptor("tableName");
4 table.addFamily(new HColumnDescriptor("f"));
5 admin.createTable(table);
6 admin.close();
7 table.close();
```

Listing 3.5: Creating an HBase table called *tableName* with the Java API

Each bitly click has its data only one time and accordingly it is no problem to set the value to one. Arguable could be the user table. If the user published a tweet then the profile data is stored. At this point the user can change the profile and afterwards post another tweet. In this case the user information will be overwritten. For analyzing values like temporal progress of number of followers, this would be a problem. But for this thesis the analysis is done regarding geographical information. Consequently it is sufficient to keep only one profile data (equal to them of the last tweet of the user in September 2013) and not several versions. This approach reduces the storage and the complexity of the queries.

The only table where the number of versions has to be greater than one, is the tweet table. One reason is the hashtag column. A tweet can contain as many hashtags as the limit of 160 characters allows. Since each hashtag needs at least two characters (one for the #-sign and one or more for the hashtag name) and each tweet has a bitly URL (needs at minimum 13 characters), the version number is set to 80. Listing 3.6 changes the number of versions of table t (line 7) to 80. The same is needed for all other HBase tables, but with the value 1 in line 7.

```
1 Configuration configuration = HBaseConfiguration.create();
2 HBaseAdmin admin = new HBaseAdmin(configuration);
3 String table = "t";
4 admin.disableTable(table);
5 HTableDescriptor td = admin.getTableDescriptor(Bytes.toBytes(table));
6 HColumnDescriptor cf = td.getFamily(Bytes.toBytes("f"));
7 cf.setMaxVersions(80);
8 admin.modifyColumn(table, cf);
9 admin.enableTable(table);
```

Listing 3.6: Changing the number of versions for an HBase table

3.2.4 Insertion of the tweets into HBase

After creating the HBase tables, it is possible to begin with the insertion of the tweets. The text files are read line by line and therefore tweet by tweet. First the tweet is converted from the *StatusJSONImpl* object to the general JSON (see section 3.2.2). Next the Java library *json-simple* is used to generate a *JSONObject*, from which all the information can be read quite straightforward. E.g. to find the identifier of the *JSONObject*, only the *get()* method is used (see line 8 of listing 3.7).

```
1 // Converting the tweet to the general form of JSON
2 String formatted = Formatter.formatStatusTweetToJSON(stweet);
3 // Initialize a JSON parser
4 JSONParser parser = new JSONParser();
5 // Parse the tweet into a JSONObject
6 JSONObject json = (JSONObject) parser.parse(formatted);
7 // Get the identifier of the tweet
8 String sid = json.get("id").toString();
9 ...
```

Listing 3.7: Usage of the Java library *json-simple* to parse a tweet into a simple readable *JSONObject*

There exist many millions of tweets to insert, so it is important to optimize the process. Instead of putting every tweet to HBase directly after loading from the file, it is advantageous to load a certain number of tweets at once to HBase. On these grounds several array lists are used (for each table one). In listing 3.8 first an array list of *Put*s is initialized. This is done before starting reading the tweets. Then for each tweet the required information are added to a Put object. Once a whole file with 100'000 tweets is read, all the Puts in the array list (line 2) are loaded to HBase (line 16).

For the first and the last file of September 2013 with streamed tweets, an additional condition is necessary. The timestamp of the tweet has to be checked if it is in the interval [1377993600, 1380585600). The start value is equal to the 1st September 2013 at 00:00:00, the end value is excluded and stands for the 1st October 2013 at 00:00:00. Because it can happen that a tweet is delayed, all tweets around these two times need to be controlled (see section 3.1.4).

```
1  // Initialize an array list of Puts to minimize the time consuming
2  ArrayList<Put> tweets = new ArrayList<Put>();
3  ...
4  // Initialize a Put with the row key of the tweet's identifier
5  Put id = new Put(Bytes.toBytes(sid));
6  // Add the timestamp to the Put
7  id.add(Bytes.toBytes("f"), Bytes.toBytes("ts"), Bytes.toBytes(String.
       valueOf(sts)));
8  // Add text of tweet
9  id.add(Bytes.toBytes("f"), Bytes.toBytes("te"), Bytes.toBytes(ste));
10 // Add user identifier
11 id.add(Bytes.toBytes("f"), Bytes.toBytes("uid"), Bytes.toBytes(suid));
12 ...
13 // Loading the tweet table
14 HTable tweet = new HTable(this.configuration, "t");
15 // Execute all puts of array list (load tweets to HBase)
16 tweet.put(tweets);
```

Listing 3.8: Put a list of tweets at once to HBase

3.2.5 Insertion of the bitly data into HBase

The data of bitly clicks provided by VeriSign is quite large. One hour of data gains a size of 5 GB on average. That is why it is important to insert only the data that is really needed for the analysis later in this thesis. In the HBase table bt, the row keys are all existing bitly identifiers of the tweets. Accordingly all bitly data, that does not contain one of these identifiers should not be taken into account.

The first approach was to verify for each bitly click if it occurs in the HBase table bt. Although the query is a *Get* and consequently very fast, it is not fast enough. One *Get*-query lasts only some milliseconds. But the amount of data is too large. There are about seven millions clicks of a bitly URL per hour on average. The first thought was, that it is the fastest way to make sure that only the wanted bitly clicks are inserted. The execution time was more than five hours to load all the needed bitly URLs of one hour to HBase. Of course it is possible to run several executions concurrent, but nevertheless it takes more than one hour per file. The total time to insert all bitly data to HBase would be more than a month. It makes no sense to load data of one month into HBase, when the loading process takes more time than the data is about.

The solution for the time-consuming problem of the first approach is to export the

bitly identifiers of the table bt into a file and load it to a tree set in Java. The advantage of this idea is to keep the list of bitly identifiers in the main memory, so that a request is much faster than always looking in HBase. Hence the loading time for the list of identifiers is needed only at the start of the program. Because of the limited memory and to avoid an out-of-memory exception the list has to be split in two parts. Although this means twice as much work for the machine as before, it is a lot faster. The duration is now about 15 minutes. If it is compared to the optimal value of one hour in the approach of looking in HBase for every bitly click (and most time it was definitively more than one hour), it is at least an improvement of factor 4. Totally, the insertion of the whole amount of bitly data is possible in one week, what is quite better than one month. There are other optimizations possible like using concurrent threads in Java, but because the topic of this thesis is not the optimization of a program's execution time, it is acceptable.

The Java implementation for the insertion of the bitly data is similar to the one of the tweets (see section 3.2.4). The two important differences are, that on the one hand the data is already in the general JSON format and does not need any conversion, on the other hand the list of bitly identifiers has to be loaded at the start of the insertion.

3.3 Clustering the inserted data of HBase by Weka

In this section the method of clustering the data based on its geographical information is discussed. Some presteps like getting all needed information out of the HBase tables or parsing the data into a format usable for Weka, are necessary.

3.3.1 Filtering necessary information out of HBase

Although it seems to be apparent to use the data for the clustering directly from HBase, this is not the case. The region servers are affected by instability problems and as soon as too many queries are done in a small time slot some of the HBase servers crash. That is why it is finally simpler and safer to export all data needed for clustering into files. Anyhow, it is inevitable to think about the exporting and especially about the storing of the data, due to the fact that the size of them can not be neglected.

Before making any decisions concerning the storage it is important to know what should be exported of HBase. The goal of clustering the geographical information is to figure out if a bitly URL occurs local or global over the whole world. Another objective is to illustrate the impact of a tweet on the geographical distribution. For this attempt, the bitly clicks before and after the publishing time of the tweet are considered and compared.

Regarding the two mentioned goals above there is different data needed from HBase. One important value is the timestamp. This value is needed for the tweet and the bitly clicks. Other inevitable values are the geographical latitude and longitude. Unfortunately, only less than one percent of the tweets possess this measurements (see figure 5.1, p. 42). On the contrary, for most of the bitly clicks these values exist. Therefore only the bitly clicks with this two values are taken into account.

Another important constraint is that the bitly URL has to be clicked frequently. Otherwise the measured results of clustering will not be representative. For this purpose a minimal limit of 100 clicks of a bitly URLs must be achieved.

Furthermore, there is one condition which was not planned from the beginning. The left chart on figure 5.2 (page 43) shows an inconsistency. The x-axis is the time of streaming (1.-30. September 2013), the y-axis shows the number of tweets measured per hour. Before the 9th September there is an irregular up and down of the total of tweets per hour. Then from the 9th September it is a visible rhythm going up during the day and down during the night. The reason is hard to find. One possibility is that the process of streaming tweets was not constant. But if the time of streaming 100'000 tweets is examined (time a file was stored, because always after 100'000 tweets it stores them to a file), there is no difference to the files of the regular rhythm. Between two files there is

always about 35 minutes difference. Another more possible reason is that the insertion was interrupted somewhere. Finally a time slot was found where some seconds no tweets are existent in the HBase table. If the insertion was interrupted for each file for a few seconds, this could explain the inconsistency. It is difficult to find all missing tweets and because there are enough data for the rest of the month, the time slot of the tweets to analyze can be shrunk to one week. That is why only the tweets of the week from the 23rd to the 29th September is taken into account. The time window for the bitly data is larger. For a tweet published on some day, the bitly clicks of one day before and after this day are analyzed and therefore the bitly data between the 22nd and the 30th September are necessary (for further explanation see section 5.3.2, p. 50).

The procedure of exporting the relevant data from HBase can be divided into the following steps.

1. All bitly identifiers with at least 100 clicks are kept. This can be done by looking at the HBase table bb. The number of clicks is equal to the number of columns, because each column in table bb is an identifier for one click. The remaining bitly identifiers are stored in a file. Totally there are 660'647 different identifiers with at least 100 clicks.

2. All tweets with a bitly identifier from the first step and whose publishing data is between the 23rd and the 29th September 2013 are exported. Only the columns *bid*, *tid* and *ts* are stored to a file. The size of the file is 126 MB and hence enough small for later processing.

3. The same conditions as for the tweets is applicable for the bitly data with a slight adaption: The time window is in this case from the 22nd to the 30th September 2013 (see explanation some lines above). The exported columns are *brk*, *bts*, *blat* and *blon*.

4. For each bitly identifier one file is created and all clicks of it are stored inside.

3.3.2 Creation of instances for Weka's analytics tools

Now the required data is exported from HBase, but before the clustering algorithm can be applied, the data has to be transformed into readable instances for Weka. Listing 3.9 shows the creation of a data set for Weka. In the first part (lines 1-8) an empty data set with the attributes *latitude* and *longitude* is created. A data set is an *Instances* object in Weka. Consequentially one data point is called *Instance*. The line 11 displays the

initialization of such an *Instance* object. Afterwards the values are set to the instance and finally the whole instance is added to the data set (lines 13-19).

```
1  // Inititalizes a fast vector
2  FastVector fastVector = new FastVector();
3  // Adds a new attribute to the fastVector
4  fastVector.addElement(new Attribute("latitude"));
5  // Creates a attribute for the longitude
6  fastVector.addElement(new Attribute("longitude"));
7  // Creates an empty Instances object with a title for the data set and
       the specified attributes, the third parameter is the capacity and can
        be set to zero at the beginning
8  Instances data = new Instances("GeoData", fastVector, 0);
9
10 // New instance with two attributes is initialized
11 Instance instance = new Instance(2);
12 // Example values for latitude and longitude
13 double latitude = 25.43;
14 double longitude = 14.12;
15 // Sets the two values for the instance
16 instance.setValue(data.attribute(0), latitude);
17 instance.setValue(data.attribute(1), longitude);
18 // Adds the created instance to the data set
19 data.add(instance);
```

Listing 3.9: Creating Weka instances from the exported HBase data

Mathematical measurements

As soon as the data set is created and all instances are loaded, it is possible to get the first measurements. With the methods *meanOrMode* and *variance* the mean and the variance of one attribute of the loaded data set is computed. Then the standard deviation can be calculated directly from the variance. The equations are given in (3.1) - (3.5) (a: attribute, i: index of instance in data set, $x_a(i)$: value of instance i with attribute a, n: total number of instances in the data set).

$$\bar{x} = \frac{\sum_{i=0}^{n-1} x_a(i)}{n} \quad \text{(mean)} \tag{3.1}$$

$$P(x) = \frac{\#(x_a(i) = x)}{n}, \quad i \in 0, 1, 2, ..., n-1 \quad \text{(probability)} \tag{3.2}$$

$$E[x] = \sum_{i=0}^{n-1}(x_a(i) \cdot P(x_a(i))) \quad \text{(expected value)} \tag{3.3}$$

$$Var(x) = \frac{\sum_{i=0}^{n-1}(x_a(i) - E(x))^2}{n} \quad \text{(variance)} \tag{3.4}$$

$$\sigma = \sqrt{Var(x)} \quad \text{(standard deviation)} \tag{3.5}$$

In Java it is rather simple to get the values. In listing 3.10 it is assumed that the data set contains the attributes *latitude*. In line 2 the methode *meanOrMode* is used. Because the latitude is made of a numeric format, the mean is calculated. If the attribute is nominal, the mode is computed.

```
1 // Gets the mean value for the attribute "latitude"
2 double mean = data.meanOrMode(data.attribute("latitude"))
3 // Computes the variance
4 double variance = data.variance(data.attribute("latitude"))
5 // Calculates the standard deviation
6 double standardDeviation = Math.sqrt(variance);
```

Listing 3.10: Java implementation for calculating mathematical measurements of a data set

3.3.3 Clustering data with Weka

For each bitly URL which was clicked at least 100 times, one tweet is taken. The timestamp of publishing this tweet is stored. Now all bitly clicks that occur the day before and after the timestamp of the tweet, are loaded. Another condition is determined: Only bitly URLs are taken into account, which occur the day before, respectively after, at least ten times. To keep the overview, all the conditions are summarized next.

1. Bitly URL has to be clicked at least 100 times.
2. Tweet appears between the 23rd and the 29th September 2013.
3. Bitly URL was clicked between the 22nd and 30th September 2013.
4. For each bitly URL only one tweet is kept.

5. Number of clicks has to be greater than 10 for one day before (resp. after) the tweet with the bitly URL was published

With these conditions the total number of bitly URLs ready for clustering is 17'809.

XMeans algorithm for clustering

Clustering demands an algorithm. Weka provides many different algorithms. After testing most of them, the XMeans algorithm finished with the best results. Another rather good one is EM (Expectation-Maximization). Both algorithms are built on the K-Means algorithm (Do/Batzoglou 2008). The EM-algorithm ends often in just one cluster if the instances are spread too much, while XMeans finds always some clusters. Furthermore, for EM the number of clusters can be specified. So if it is set to 2, the data set is divided in two clusters. If the value is -1, it automatically finds the clusters, but in many cases only one cluster. For the XMeans algorithm the minimal number of clusters can be adjusted. That is the reason why XMeans is favored.

The K-Means algorithm has three essential disadvantages. First, the number of clusters (K) has to be specified by the user, thus it is not able to find the optimal number of clusters for a data set. Second, the results depend on the choice of points by the user at the beginning (Loss 2002, p. 5). The third disadvantage is that the computation takes a lot of time for larger datasets. These three properties are improved by the XMeans algorithm. The number of clusters and the start points are estimated automatically and the algorithm is optimized concerning speedup by using blacklists, so that not every point needs to be considered. (Pelleg/Moore 2000)

The usage of the XMeans algorithm in the Java API of Weka is displayed in listing 3.11. In the lines 2–8 the XMeans object is instantiated and some properties like the minimal or maximal number of clusters are set. With the *buildClusterer* method the clustering algorithm is executed. Two more important methods are listed. The first is *getClusterCenters*. This method returns all the centers of the found clusters (as *Instances* object). The *clusterInstance* method outputs the index of the found cluster. It would be interesting to know which point belongs to which cluster or to get the number of points per cluster. If the method *clusterInstance* is applied to the whole data set, new ones are build containing all instances of a cluster (and if wanted, running the clustering algorithm again to this smaller data sets). Another benefit of creating these subsets is to be able to calculate measurements like mean, variance or standard deviation for each cluster. Here is to mention that only clusters with at least three elements are kept for the analysis of the results.

Furthermore the XMeans algorithm calculates a distortion value, which stands for the quality of the clusters. The lower the distortion is, the more representative is the cluster.

```
1  // Initializing an XMeans object
2  XMeans xmeans = new XMeans();
3  // Raises the maximal number of iterations from default 1 to 10
4  xmeans.setMaxIterations(10);
5  // Minimal number of clusters is 2
6  xmeans.setMinNumClusters(2);
7  // Maximum number of clusters is 10
8  xmeans.setMaxNumClusters(10);
9  // Applies the XMeans algorithm to the data set
10 xmeans.buildClusterer(data);
11
12 // Gets the centers of the clusters
13 Instances centers = xmeans.getClusterCenters();
14 // Returns the index of the cluster a specific instance has
15 int clusterIndex = xmeans.clusterInstance(instance);
```

Listing 3.11: Applying XMeans algorithm in Weka to a data set

Output of the clustering with XMeans applied to a bitly URL

The filtered data which fulfills the five conditions defined at the beginning of this section, are clustered by the XMeans algorithm with the properties set in listing 3.11. A lot of measurements are made and stored to a file. An example of the output is given in the tables 3.3 and 3.4. The first table shows the different measurements directly calculated from the data set. The table 3.4 contains the output of applying the XMeans algorithm to an example data set. Here is to mention, that for each cluster it is possible to measure the parameters of 3.3 as well.

Measurement	**Value**
Elements	51
MeanLatitude	22.95776874
MeanLongitude	67.48595952
VarianceLatitude	55.16353824
VarianceLongitude	1755.354383
StandardDeviationLatitude	7.427216049
StandardDeviationLongitude	41.89694956

Table 3.3: Measurements of the data set with the example bitly identifier "104QQyK"

Cluster	**Measurement**	**Value**
	Distortion	1.048861
cluster0	CenterLangitude	42.13093313
cluster0	CenterLongitude	-96.25606537
cluster0	Elements	3
cluster0	ElementsOfTotal	0.05882353
cluster1	CenterLangitude	28.50621455
cluster1	CenterLongitude	77.10739844
cluster1	Elements	14
cluster1	ElementsOfTotal	0.27450982
cluster2	CenterLangitude	22.45456653
cluster2	CenterLongitude	86.17754279
cluster2	Elements	9
cluster2	ElementsOfTotal	0.1764706
...	...	...

Table 3.4: Measurements returned after executing the XMeans clusterer to the example bitly identifier "104QQyK"

These measurements are saved in a file, imported to a calculation program like Microsoft Excel and analyzed. The results are shown in the chapter 5.

Chapter 4

Experimental Architecture

In this chapter the used technological data can be found.

4.1 Machines

Four machines are used for the Cloudera cluster. They have all the same components:

Processor: Intel(R) Core(TM) i7-3770 CPU 3.40GHz
RAM: 4096 MB
Hard Drive: 500 GB
Operating System: Linux Ubuntu 12.04
Java: Version 1.0.7 Update 45 (although HBase advises version 1.0.6, all operations which need Java worked without problems)

4.2 Cloudera's Distribution Including Apache Hadoop (CDH)

For the cluster setup of three machines (see section 4.1) with distributed HBase the CDH is used. Version 4.5.
Subcomponents:

- Cloudera Manager Agent: Version 4.8.0
- Hadoop: Version 2.0.0
- HDFS: Version 2.0.0

- Zookeeper: Version 3.4.5
- HBase: Version 0.94.6
- IMPALA: Version 1.2.3
- SOLR: Version 1.1.0

4.3 Java Libraries

The following Java libraries are used during streaming tweets, inserting them to HBase and analyzing.

- twitter4j: For an easier access to the Twitter components and functions. Primarily this library is used for streaming tweets. Version: 3.0.5.
- hbase: For interactions with the Cloudera cluster, especially with the HBase tables. Version 0.94.9.
- weka: The weka-library is useful for analyzing, clustering and classifying data. Version 3.6. Furthermore there exists an explorer with a visualization tool to try out small samples of data (see section 2.4.1). This explorer is a very helpful application before starting to use the weka Java API. It gives a good overview of the functionalities and its possibilities.
- json-simple: To parse JSON-formatted files. Version: 1.1.1.
- stanford-ner: For named-entity recognition in tweets. Version: 3.2.0.

Chapter 5

Results and Discussion

In this chapter the results are presented and a lot of statistics will be discussed. First general measurements of the HBase tables are analyzed. Afterwards some examples with the bitly URLs are made und finally the impact of a tweet on the geographical distribution of the bitly clicks is considered.

5.1 General statistics of the HBase tables

Before interpreting any charts, it is good to know the size of the HBase tables. The total number of rows for each table is shown in table 5.1. The most entries has the bitly table b with 155 millions, storing all the information of the clicks on a bitly URL. Further the tweets table t is rather large too, with 90 millions rows. Totally 4.7 millions Twitter users are registered. For the place information only 22'807 are found. While there exist for 84% of the bitly clicks the latitude/longitude values, only 0.4% of the tweets table contain this parameters (figure 5.1). The latitude/longitude is stored in the tweet table t itself, but it shows that there are not many geographical information for the tweets and explains the small amount of different places.

HBase table	**Number of rows**
t (tweets)	90'617'305
u/ut (users)	4'721'299
p/pt (places)	22'807
b (bitly clicks)	155'378'302
bb (bitly ids in clicks)	13'104'428
bt (bitly ids in tweets)	41'558'647

Table 5.1: Total number of rows for each HBase table

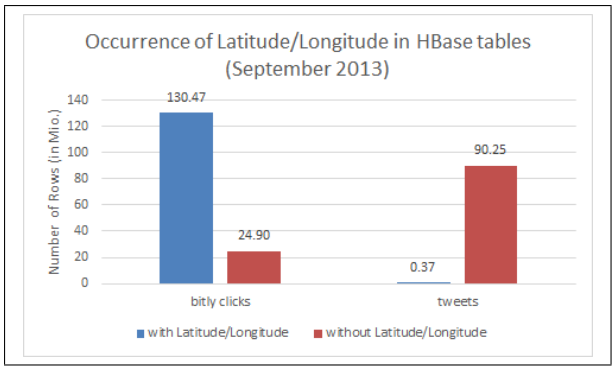

Figure 5.1: Occurrence of the geographical parameters latitude and longitude in the HBase tables of the tweets and bitly clicks during September 2013

In figure 5.2 there are two charts illustrated. The left one presents the total tweets posted during September 2013. They are summarized for each hour. The inconsistency of the first week of September is discussed in section 3.3.1, where the reason is not clarified absolutely. The most probably cause is that the insertion respectively the HBase region servers were instable, because for some seconds there exists no tweets in HBase. There is a noticeable peak during the 11th September in the right chart with the number of bitly clicks per hour. The number of clicks reaches new dimensions on this day. Already at 2 o'clock a.m. 604'377 clicks are registered. At 8 o'clock p.m. the absolute top value of 986'524 clicks is measured. A possible motive is the Patriot Day in the United States. During this day the people remember those who died during the terroristic attacks to the World Trade Center in New York (Timeanddate 2014). The constant increase over time of the total bitly clicks occurs, because i.e. on the 10th September there are all bitly identifiers of this day and the nine days before given. Now if the 20th September is considered, the new identifiers of the days between the 10th and 20th September are added too.

In both charts the daily routine is visible. The values fall in the night and are obviously higher during day. There is one more difference between the charts. The peaks per day are for the Twitter data around lunch or afternoon, while the bitly URLs were clicked more in the evening. The reason could be that bitly URLs are often linked to audio-visual materials like movies. They need more time to watch, than just reading a message of a tweet. During work it is easier to post a tweet than looking a whole movie.

If a look is taken on the figure 5.3, a constant number of about 3 millions tweets per day of week is shown in the left chart (only a small decrease on Sunday). Here is to comment that the values are not the total number of clicks but the average. This is important because the month September 2013 began with a Sunday and ended on a

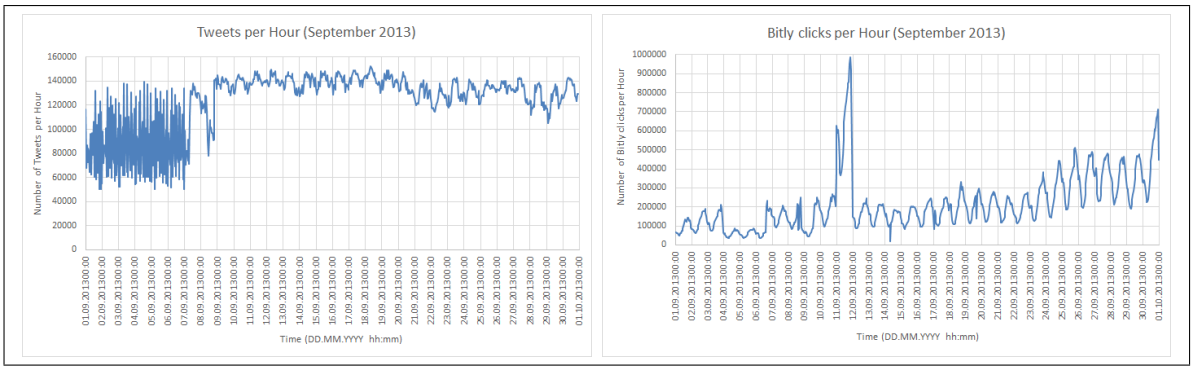

Figure 5.2: Total number of tweets (resp. bitly clicks) counted per hour during September 2013

Monday. Therefore, these two days occur one time more than the remaining and the average is needed instead of the total. For the bitly clicks, Wednesday has by far the top value. The reason is again the 11th September, when more than 15 millions clicks were made and therefore 10 millions more than on the other Wednesdays. If this day is removed, an average of 4.84 millions is calculated, which is in the range of the other days. Again the lowest value is reached on Sunday. Summarizing it can be said, that the activity on Twitter is on Sunday the lowest.

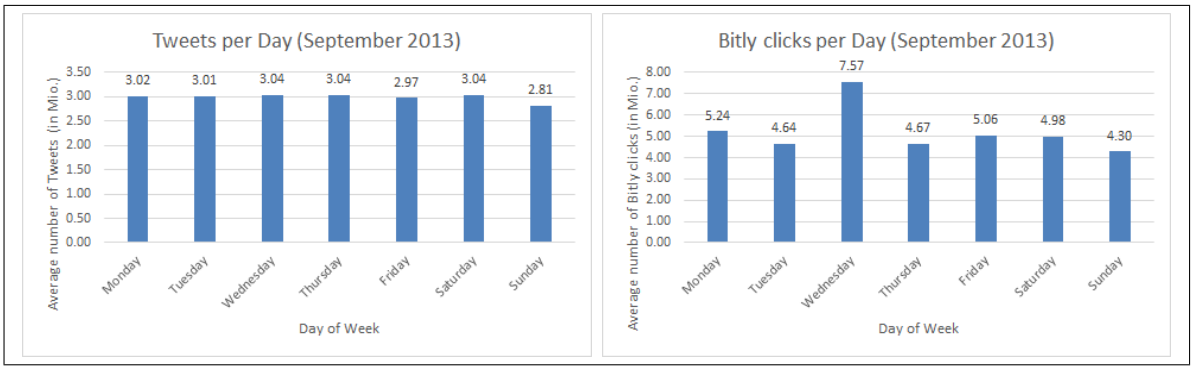

Figure 5.3: Average number of tweets (resp. bitly clicks) per day of week during September 2013

In the two charts of figure 5.4, the total tweets (resp. bitly clicks) are counted per hour for the whole data set of September 2013. Here is to mention that the y-axis have not the same ranges and intervals. While there occur between 3.2 and 4.2 millions tweets per hour, bitly clicks are mostly higher. Especially the difference between day and night is considerably more visible for the bitly clicks with the peak at 20 o'clock (20:00–20:59) with more than 9 millions clicks. Interesting is that most tweets are posted at 10 and 15 o'clock. The cause could be the breaks during work. Other high values are reached during breakfast and lunch. The reason, why there are many tweets published in night at

1 o'clock is unclear. Maybe some automatically generated tweets (like spam) are started at midnight.

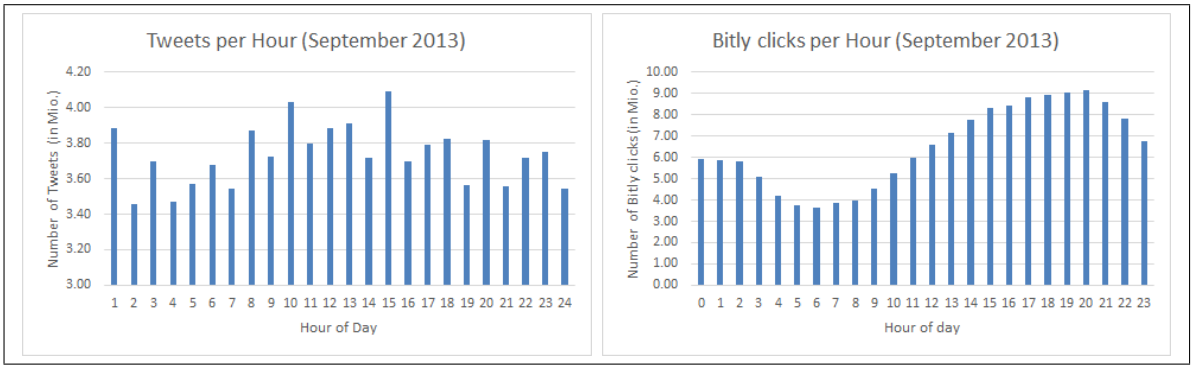

Figure 5.4: Number of tweets/bitly clicks published during September 2013 ordered by the daytime

5.1.1 Occurrences of hashtags

From the tweets table the hashtags can be counted, leading to a very surprising result. At the top of the ranking in figure 5.5 is unexpectedly the hashtag #smurfsvillage with an occurrence of 643'974 times during the month September in 2013. In disbelief the look on Twitter searching for the hashtag #smurfsvillage really returned several tweets per minute and each tweet contains a bitly URL. The comparison with the second most occurring hashtag #news on Twitter confirmed the values. Although there are more tweets published with the hashtag #news, a lot of them have no bitly URL. The list shows only understandable hashtags. Asian or Arabic hashtags are not listed in figure 5.5. If the objectives are compared, there are a lot of hashtags concerning mobile phones (iphone, android, ebaymobile), gaming (smurfsvillage, win), social media (fb, socialmedia), business (job(s), marketing) or sexual content (freeporn) in the top-rankings.

Rank	Hashtag	Occurences
1	smurfsvillage	643974
2	news	396931
3	pandora	280413
6	jobs	227063
10	followback	154462
11	followme	147048
15	iphone	120186
16	nowplaying	111639
18	job	93436
27	football	71113
28	android	68674
31	fb	65854
32	socialmedia	62793
43	win	49553
44	marketing	49514
47	freeporn	49208
48	apple	49158
49	ebaymobile	47981
286	obama	10371
242	usa	11809
624	schweiz	5442
6748	switzerland	464

Figure 5.5: Ranked occurrences of hashtags during September 2013

5.2 Impact of Tweets on the number of Bitly clicks

Before continuing with the geographical impact of a tweet, it is important to show that the number of bitly clicks is influenced by the tweet. To get a measurable parameter, the number of bitly clicks before (variable b) and after (a) posting a tweet are considered. Instead of just taking the normal ratio (i.e. b/a) and getting a wide-spread chart, which is harder to analyse, the ratio should be adapted for this purpose. The new computed ratio is the difference of the number of elements divided by the sum (see equation (5.1)).

$$impact = \frac{a-b}{a+b}$$
$$\text{where } a \text{ is the number of bitly clicks after posting the tweet} \tag{5.1}$$
$$\text{and } b \text{ before posting the tweet}$$

In figure 5.6 the defined impact is the x-axis and the number of bitly identifiers is the y-axis. For many bitly identifiers there is no change in the number of bitly clicks (impact=0). If the positive and negative ranges are compared, it is recognizable that the positive area is larger respectively there are more bitly identifiers. This implies that the number of bitly clicks increases after posting the tweet and therefore an existing influence of the tweet to the number of bitly clicks is shown. This fact is useful, because when there is an impact on the number of clicks then an impact on the geographical distribution is probable as well, which will be discussed in the next section.

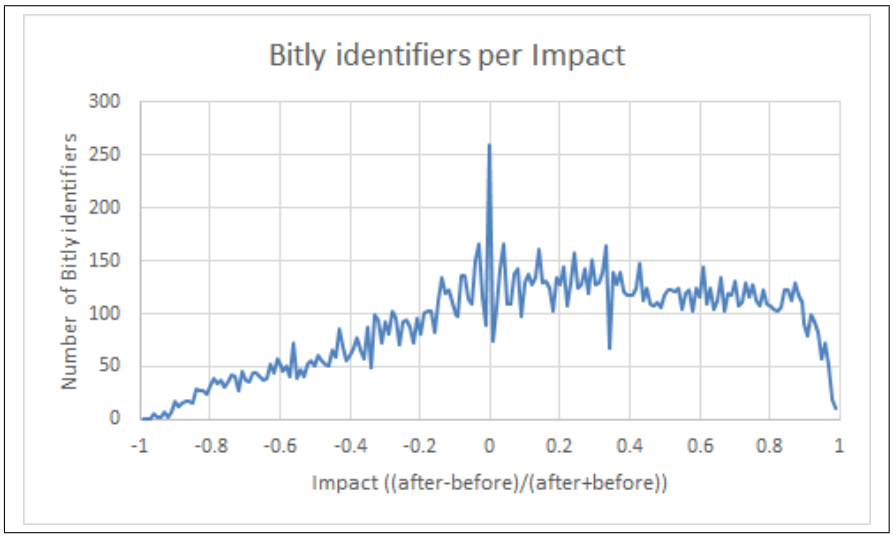

Figure 5.6: Impact of tweets on the number of bitly clicks

5.3 Geographical results

This section explores the impact of a tweet on the geographical distribution of bitly clicks.

Although there exists geographical information in the HBase tables like time zones, country code or city names, the examination is done by using the latitude and longitude values of the bitly clicks. There are several reasons why the analysis is done on these two parameters. On the one hand, the latitude and longitude return the most exact geographical position. On the other hand, most of the tweets have only one geographical value: The timezone (the UTC offset can be derived from the timezone). In the profile of the Twitter users it is possible to enter a location. But the input is a text field and everything can be stored. Often the user wrote some none interpretable locations (i.e. hearts before and after the location to symbolize for everybody looking at the profile how much he/she likes this location). Therefore, this location textfield can not be kept as a representative geographical information. Furthermore, the timezones of Twitter are specified with different names than the ones of bitly. So they must be compared and merged. Nevertheless, the concerning geographical area stays a large region and is definitively not same precise as the latitude and longitude. For the most bitly clicks the country is given. But if two countries are brought face to face (i.e. USA and Switzerland), the geographical size differs extremely.

There is one disadvantage present in the data. Because the impact of a tweet on the bitly clicks should be measured, it would be useful to have the position of the tweet. Unfortunately as described above, most of the tweets have only the timezone as geographical representative, which is too large for specific measurements on the distribution. Some tweets have the parameter latitude and longitude, but only 0.4% (see figure 5.1). These are 367 thousands tweets, which still sounds to be much. But by looking more precise to the table 5.1 with the number of rows per table, there is a large difference in the number of bitly identifiers. The table *bt* contains all the bitly identifiers that occur in at least one tweet as row key. The same holds the table *bb* with the difference that all bitly identifiers of the clicks are stored. Totally more than 28 millions bitly identifiers are never clicked during September 2013. In percent this is 68%. Additionally, there are 12.4 millions identifiers, which were clicked less than one hundred times. That means that they are not taken into account for the analysis in Weka, because the deep number of clicks is not enough representative for any conclusions. Therefore another 30% to ignore. If the assumption is made that the latitude/longitude values are distributed uniformly over all tweets, totally 98.4% will be ignored, because of having too less data for clustering. Finally there are only 6 thousands tweets that could be analyzed and have a latitude

respectively longitude value. There are more conditions (see 3.3.3, p. 35) like "there has to be at least ten clicks one day before and after the publishing time of the tweet" and hardly any tweet with latitude and longitude will survive this condition. Hence the geographical position is not assignable at the moment for the tweet. Later it will be shown that for some tweets the location can be estimated approximatively by analyzing the cluster centers of the bitly clicks before and after the publishing time of the tweet.

5.3.1 Overview of the whole amount of geographical information

On the world maps in figure 5.7 all latitude and longitude values of the tweets and bitly clicks are summarized. The more tweets (resp. bitly clicks) occur in a latitude/longitude coordinate the darker is the point colored. It is very interesting how good the continental shapes match with the given world map. The advantage that the humans live denser along the coasts could be exploited to draw a world map just by coloring all coordinates where a tweet or bitly click exists.

The top chart is generated by all tweets with the mentioned parameters. The coloration is set to be exponential from light grey over grey to black. This means that if only one tweet occurs on a certain latitude/longitude value it holds the color light-grey. If there are fifty tweets the coordinate gets grey. The more tweets are available the darker is the coordinate and from 2500 tweets the field is black. For the bitly clicks the settings of the colors are different. The mean parameter of the color grey is set ten times higher. This implies to the color grey for 500 and black for 250'000 bitly clicks. Light-grey remains on the one-time occurrence.

In both charts Europe, North America and Japan have the highest density. In South America, Africa and Australia the occurrences are concentrated along the coasts. Logically the Sahara stays nearly blank (and of course the Arctic and Antarctic). The chart of bitly clicks (middle) seems to have a more important role in Brasil and India than the Twitter data. The opposite seems to be the case for the Arabian Peninsula. Although some black dots are visible, most of the peninsula stays blank, while the tweets appear on a large region. Another interesting area in the map of bitly clicks is around the English Channel between Great Britain and France. Where the sea should be, all is grey. Either the people do not turn off their mobile phones in the aeroplanes for the flight from Great Britain to Central Europe or backwards, or there is a high workload of the maritime traffic.

Another impressive figure is visible in figure 5.8. The number of bitly clicks is summarized per latitude (chart right) and per longitude (chart bottom). Even if the world

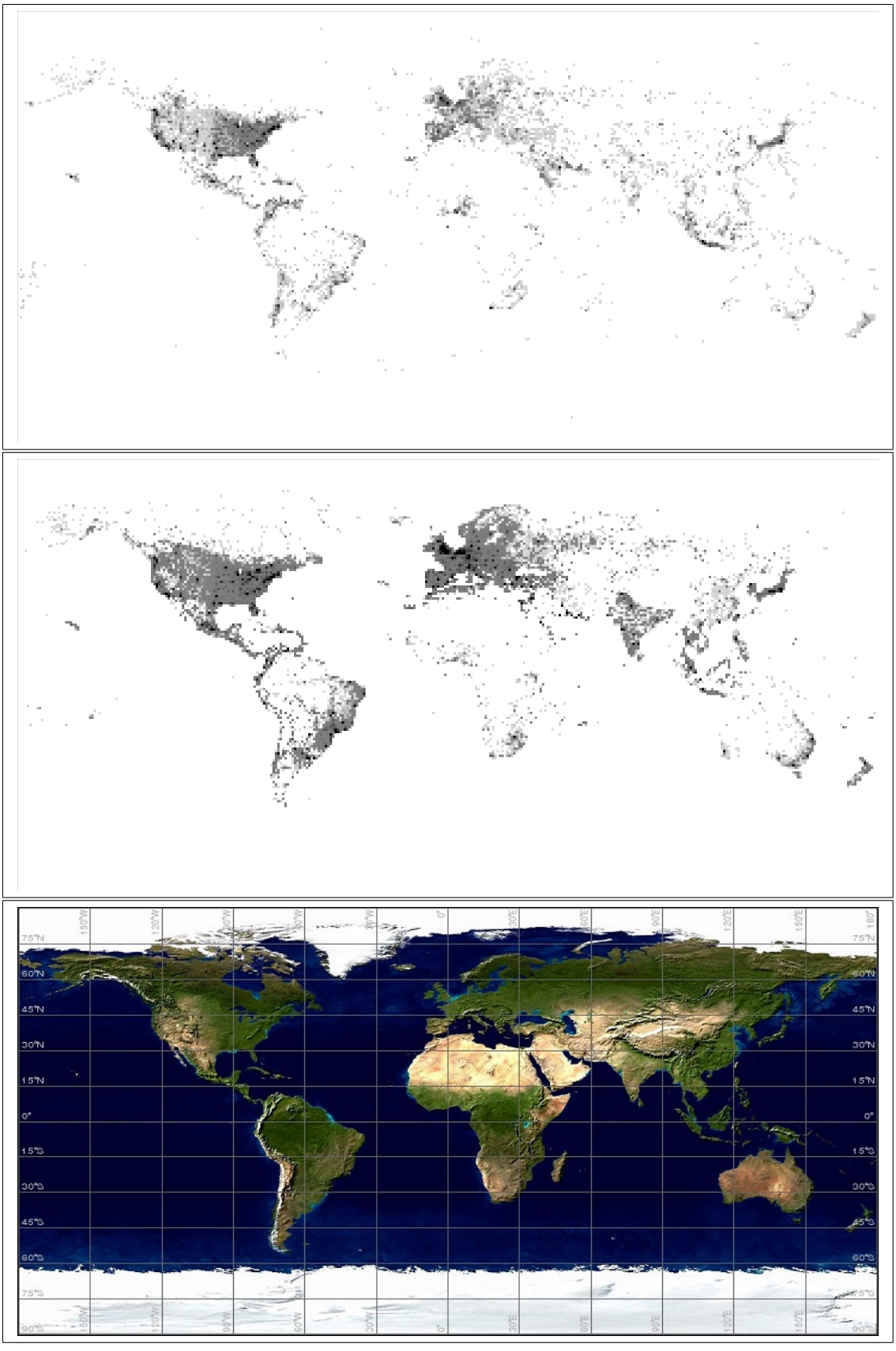

Figure 5.7: Occurrences of latitude and longitude values in tweets (top) and bitly clicks (middle) spread over the whole world. The bottom world map is for comparing (source: (Generic Logic 2014)).

map would not be there for comparison purpose, it would be possible to detect most of the continents anyhow (for the longitude; for the latitude it could get more tricky). Especially Europe and America is identifiable. For the longitude there is one absolute peak at -74 degree. New York is the reason, which lies on this longitude. Other top values exist for -47 (Brasilia), 0 (London), 29 (Istanbul) and 140 (Tokyo). For the latitude there is a difference between the northern and southern hemisphere recognizable. The bitly clicks in the first mentioned hemisphere occurs definitively more often than in the southern. Again some top latitude values are measurable. With nearly eight millions clicks the latitude 41° achieves the highest value. Like before it is the latitude of New York. In addition Madrid, Istanbul and Peking are other large cities at this latitude. The second highest value belongs to London and several other European cities like Amsterdam and Berlin with the latitude 52° and more than five millions clicks.

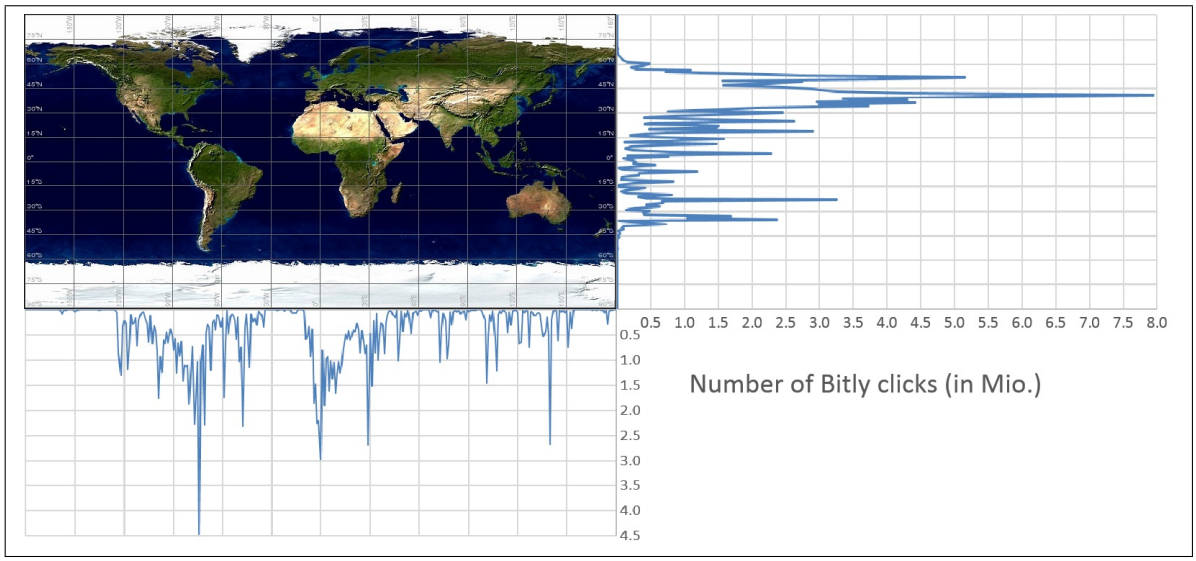

Figure 5.8: Bitly clicks per latitude (right) and longitude (bottom). (World map's source: (Generic Logic 2014))

5.3.2 Results of XMeans clustering algorithm of Weka

The figure 5.9 explains which bitly clicks are taken into account for the Weka clustering algorithm. In the center of the time axis is the timestamp of the published tweet. To both sides of this point one day is added (resp. subtracted), so that a time interval of two days is generated. This two-day segment is the accepted time for the bitly clicks. Only clicks with an absolute timestamp difference to the posting point of the tweet less than 86'400 (total seconds per day) are accepted (see equation (5.2)). Of course the bitly identifier

has to be the same for the tweet and the bitly clicks. The colored marked points on the time axis are some example clicks. Blue are the accepted clicks during the day before the tweet was posted; green the day afterwards and red all the unacceptable ones, which will be ignored for clustering.

$$|timestamp_{tweet} - timestamp_{bitlyClick}| < 86400 \tag{5.2}$$

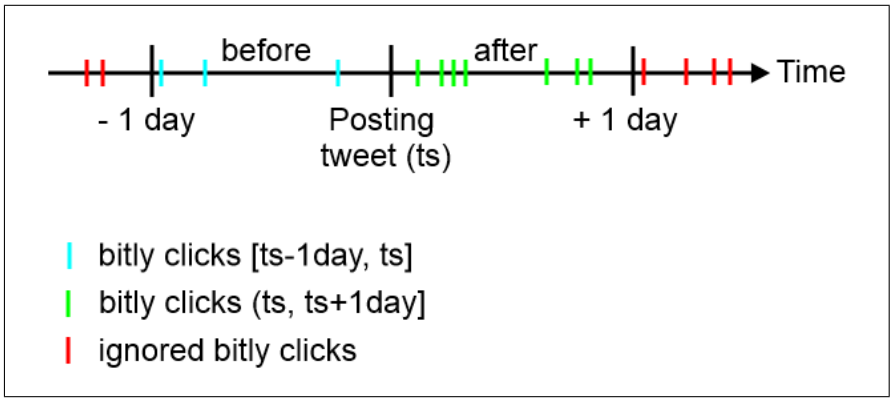

Figure 5.9: Bitly clicks spread over the time axis. Green and blue clicks are taken for clustering, red ones are ignored.

Totally 17'809 bitly identifiers are clustered with the XMeans algorithm of Weka. For each of them XMeans was executed three times:

- All bitly clicks one day before and after the tweet.
- All bitly clicks one day before the tweet.
- All bitly clicks one day after the tweet.

Therefore, the clustering algorithm was applied to 3·17'809=53'427 different data sets.

One important measurement of XMeans is the distortion. This parameter calculates how much the instances of a cluster are spread. For each computed cluster the center coordinates are available. To get the distortion the clusterer measures the squared distance of each instance in a certain cluster to the center and summarizes them. All totals of the clusters are summarized too, which results in the distortion value. The equation (5.3) shows the formula for the distortion. $d(x_{k_i}, c_k)$ is the squared Euclidean distance of an instance x_{k_i} in cluster k to its cluster center c_k. n_k is the number of instances in cluster k. For each cluster at least three elements exist, otherwise the cluster is not taken into the analysis, because only one or two bitly clicks are not enough representative for defining a cluster. (Pham/Dimov/Nguyen 2004)

$$
\begin{aligned}
distortion = & \sum_{k=1}^{\#clusters} \sum_{i=1}^{n_k} d(x_{k_i}, c_k)^2 \\
\text{where } d(x_{k_i}, c_k) = & (x_{k_i}(latitude) - c_k(latitude))^2 \\
& + (x_{k_i}(longitude) - c_k(longitude))^2
\end{aligned}
\tag{5.3}
$$

It can be said, that the distortion shows how compact the clusters are. Applied to geographical information, the distortion shows if the bitly clicks occur global or only local. The higher the distortion is, the more distributed are the bitly clicks (resp. the more global).

Figure 5.10 shows the three distortion curves. Two curves display the distortion resulting from applying the clustering algorithm XMeans to the bitly clicks before respectively after the tweet was published. Additionally, one curve is generated by the clusterer looking on all bitly clicks in the two-day interval (bitly clicks the day before and after posting the tweet). The y-axis measures the number of bitly identifiers with a certain distortion value. There is one remarkable point in the chart at a distortion between 5 and 6: All three curves intersect each other. Before this point, the number of bitly identifiers is highest for the distortion before the tweet was published and the total curve is lowest. Afterwards all three curves change their position and the distortion of all bitly clicks is highest and the ones before the tweet lowest. If the comparison between the clicks before and after posting the tweet is made, it is visible that there are more bitly identifiers with a high distortion afterwards. Only before the intersection point, the number of bitly identifiers is higher before the tweet was published. This can be interpreted, that the tweets make the bitly clicks more global.

Distortion difference of the clusters

After computing the distortion for the clustering of the bitly clicks before and after the tweet, it is possible to compute the difference (see equation (5.4)). Here is to notice, that the difference can be positive or negative. The absolute value would distort the measurements. On the one hand if the distortion of the bitly clicks after posting the tweet is larger than the ones before, the distortion raises into the positive range. On the other hand if the opposite is the case and the distortion of the bitly clicks before posting the tweet is larger, then a negative value is received. If now the absolute value is taken, the two very different cases obtain the same value and a representative analysis would not be possible any more.

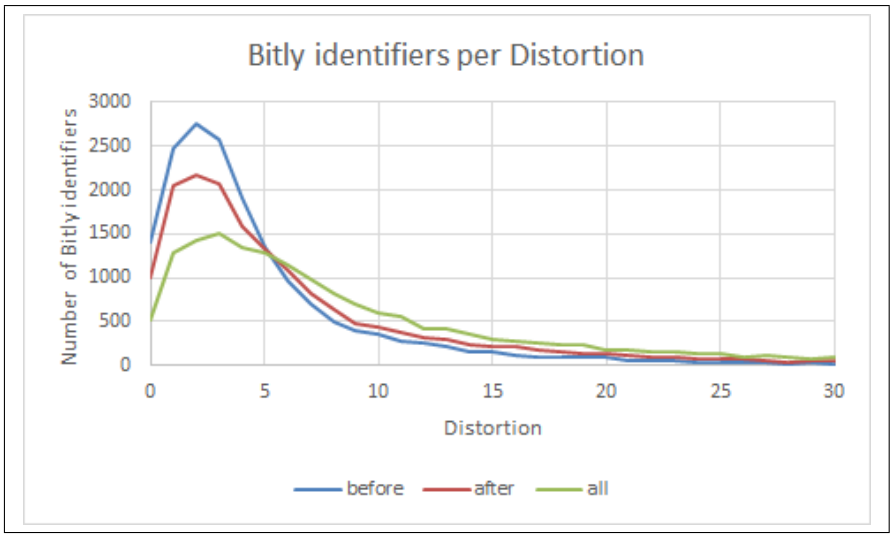

Figure 5.10: Number of bitly identifiers for the three distortion measurements of the clustering algorithm XMeans

$$distortionDifference = distortion_{after} - distortion_{before} \tag{5.4}$$

The distortion difference is an attempt of having a measurement that shows the impact of a tweet to the geographical distribution of the bitly clicks. The best way to check the influence of a tweet can be done by looking at several examples of bitly identifiers with high and low distortion differences.

In figure 5.11 several varying distortion differences are displayed. On the left side is always the geographical distribution of the bitly clicks before posting the tweet. The right map stands for the bitly clicks after the tweet. It is obvious that for small distortion differences the geographical distribution is local. For the examples with the measured values of 50 and 335, it is the opposite: The occurrences appear more global. Therefore, the impact of a tweet is especially recognizable as soon as the distortion difference takes on a large value.

The examples in figure 5.11 show only non-negative distortion differences. But this measurement could be in the negative range too. The results are in contrast to the ones with high positive differences. Instead of wide-spread geographical occurrences after posting the tweet, it gets more local clusters. Figure 5.12 displays an example of a bitly identifier with a distortion difference of -77. The number of bitly clicks is decreased. One reason could be that the bitly URL was published somewhere else than on Twitter and was clicked many times. Afterwards a Twitter user posted a tweet with the URL, but the user itself is insufficient popular to influence the number of bitly clicks. For most of the

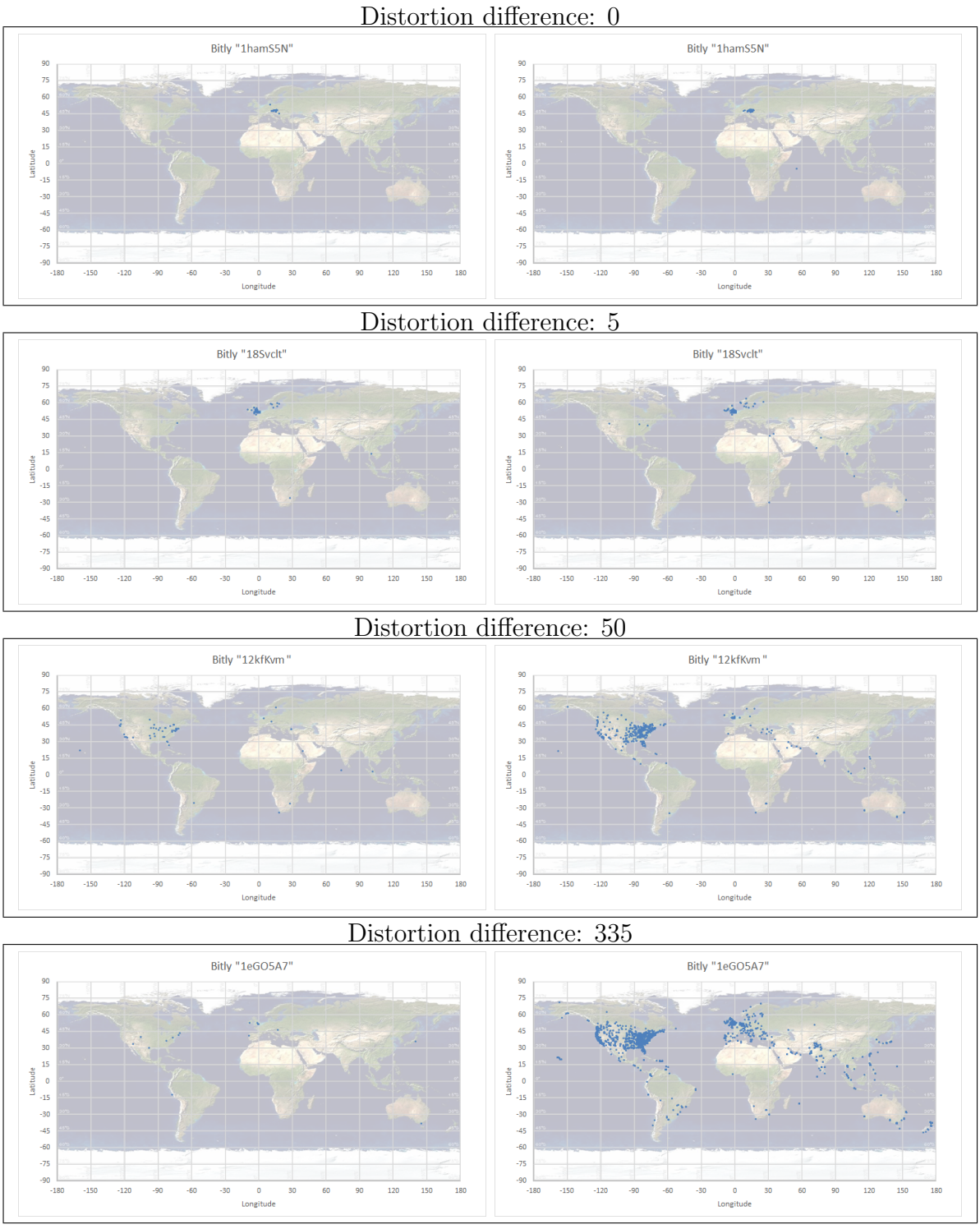

Figure 5.11: Comparison of non-negative distortion differences (left map: bitly clicks before posting the tweet, right map: afterwards) (Generic Logic 2014, the background world map's source for this and following examples).

bitly identifiers where the distortion difference is negative, the tweet looks like having no recognizable impact on the bitly clicks.

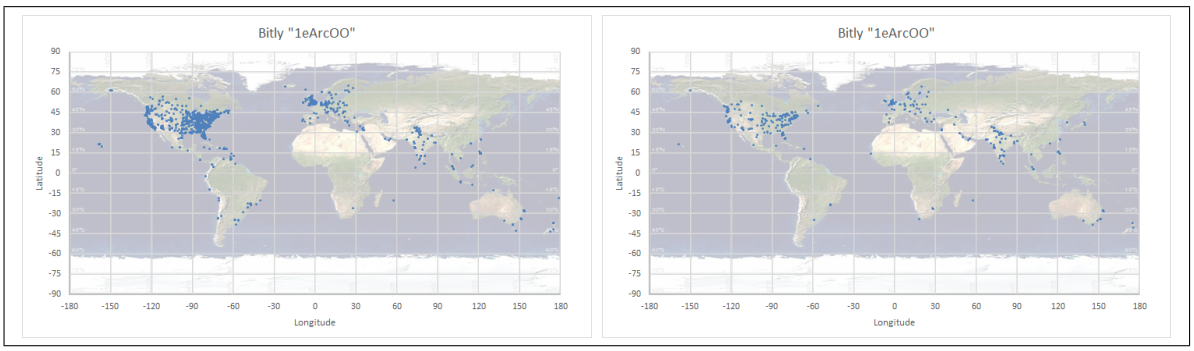

Figure 5.12: Negative distortion difference (left: before tweet, right: after tweet)

In figure 5.13 the difference of the distortion is visible on the x-axis. The y-axis represents the number of different bitly identifiers in total. If the x-values around ±5 and ±10 are compared, it can be said that there are about two times more bitly identifiers with a positive distortion difference than a negative. Of course for the highest number of bitly identifier the difference is around 0. This is mostly the case for tweets with a low impact on the geographical distribution. There are not many bitly identifiers left with a distortion difference higher than 20. But as shown in the example of figure 5.11 with a value of 5, already a small spreading is recognizable.

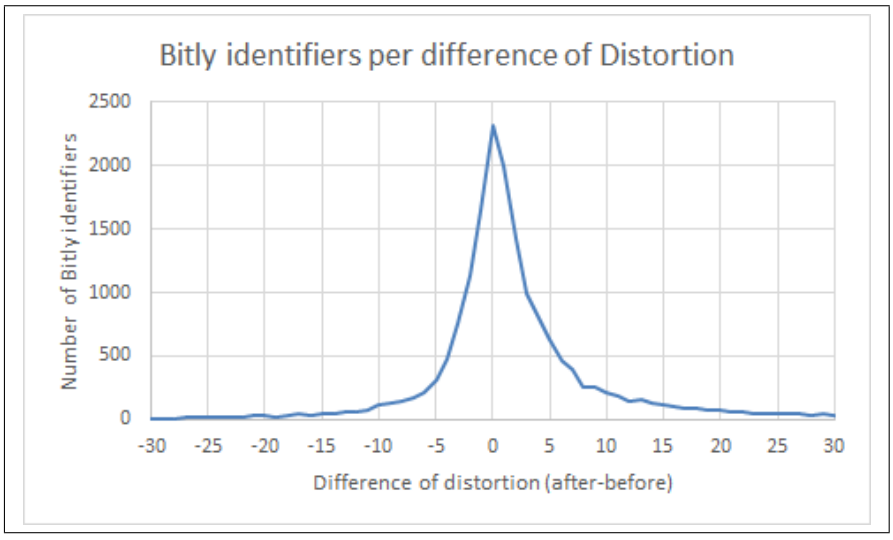

Figure 5.13: Number of bitly identifiers distributed on the difference of the distortion of the XMeans clusterer applied on the clicks before and after the published tweet.

At this point it is to mention that in the choice of the data only bitly identifiers are taken, which already occur before a tweet. For all those who publish a new created bitly

URL, no clicks are available before and therefore no comparison can be made. But for these tweets an impact on the geographical distribution is present and a lot of Twitter user just create a new bitly URL each time. This is one reason why there are not many tweets with a large impact.

Furthermore, a lot of bitly identifiers are clicked many times before the tweet is posted and therefore are already wide-spread over the world. Even if the Twitter user has a very high popularity and many followers, it influences the clustering only insignificant.

Another problem is the size of countries. The United States are in terms of surface area huge compared to a country like Switzerland. A tweet published in the United States is geographically spread over a larger area than in Switzerland. Moreover the world is not a regular easy-measurable area. I.e. the sea or the Sahara restricts the usage of Twitter respectively bitly URLs. That is why an impact of a tweet on the geographical distribution of the bitly clicks is often visible, but it is hard to gain a representative model because of the irregularity of the world's geographical shape. For each coordinate of latitude/longitude it would be necessary to develop an independent model, for what the available data is not sufficient.

5.4 Named-entity recognition

Although there is no geographical influence available for this section and it is out of this thesis' scope, the results of a performed named-entity recognition on an amount of tweets should be given and discussed here briefly. Related and more specific approaches can be found in (Li et al. 2012) and (Ritter et al. 2011).

Several libraries, which support functions for named-entity recognition (NER), are applied on a small sample of fifty tweets. The reason is the adapted language of the Twitter users because of the character limitation of the tweets. A lot of abbreviations are used and therefore it is more difficult to do NER on tweets than on usual written text.

The Stanford-NER is one of the most popular applications used for named-entity recognition (Stanford 2014). There are several libraries with different number of classes. One model of them consists of the 4 classes: "Location, Person, Organization, Misc". In the 7-class library are some more numerical objectives: "Time, Location, Organization, Person, Money, Percent, Date". In the table 5.2 the precision and recall measurements are displayed for the different libraries. The default Stanford library has 3 classes. Because of the special language on Twitter, in one attempt all hashtags signs are removed, otherwise the hashtags are never recognized to a class. Further the Stanford library was applied after deleting all special signs (.,#@ etc.). The Heideltime library only finds time-specific entities in the tweet and was adapted accordingly (Strötgen/Gertz 2010).

The precision and recall values can be explained the best way by an example. For the default Stanford the precision is 0.93 and the recall 0.42. This can be translated into "42% of all entities were found, whereof an entity was correctly with 93% probability". The goal is to get both values as near to 1 (100%) as possible. In figure 5.14 the values are displayed graphically. Heideltime seems to be the best one, because it is nearest to the optimal 1/1 coordinate. But as mentioned above Heideltime considers only time-specific parameters. Most of the libraries resumed nearly similar results. Stanford with 4 classes has a good recall, but in exchange a bad precision. For the others it is the opposite. Quite good is Stanford after removing all signs, that recognized definitively more entities than the usage of original tweets.

Library	Precision	Recall
Stanford (Stanford 2014)	0.93	0.42
StanfordWithoutHashtagSign	0.93	0.46
StanfordWithoutSigns	0.90	0.56
Stanford4Classes	0.64	0.64
Stanford7Classes	0.92	0.47
SMILE Text Analyzer (SMILE 2014) (online)	0.88	0.24
Heideltime (HeidelTime 2014) (only time)	1.00	0.80
AlchemyAPI (AlchemyAPI 2014)	0.93	0.44

Table 5.2: Named-entity recognition applied by several libraries on 50 tweets

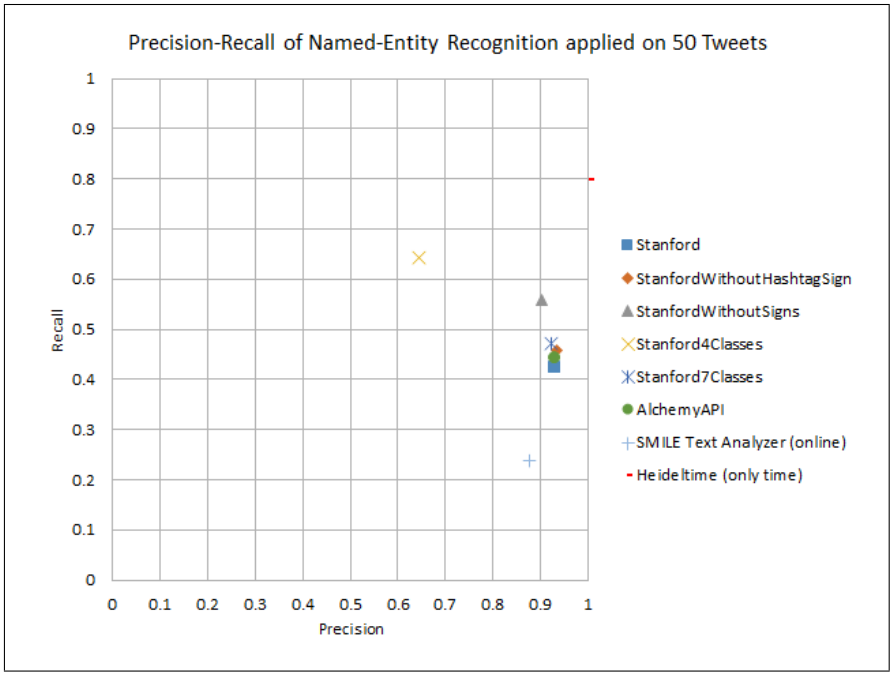

Figure 5.14: Named-entity recognition applied on a sample of 50 tweets measured concerning precision and recall

Chapter 6

Conclusions

In this work it was possible to show that tweets impact bitly clicks. On the one hand they increased the number of clicks. On the other hand they influence the geographical distribution too.

The number of analyzed bitly identifiers, which were clustered with Weka, is with 17'809 enough high to get a representative amount of data. But with a total of about 13 millions different bitly identifiers inserted into HBase, only a small fraction fulfilled all defined conditions. The major part of these clustered bitly identifiers have hardly any noticeable shift for the geographical distribution. The reason is that there are so many bitly clicks before the tweet was published, that the influence is not visible in the large amount of data. After posting the tweet, the geographical distribution is often almost the same.

Are there not too many bitly clicks before the tweet, the impact of a tweet can be examined best. Especially with the distortion difference computed by the XMeans algorithm of Weka it is possible to show the influence and the change for the geographical distribution.

One great problem is to find a general model for the impact. Because of the irregular world map with a lot of sea, where hardly any bitly clicks respectively tweets occur, the expansion is for each coordinate (latitude and longitude) different and a generalization is difficult to realize. A tweet posted in a coastal city influences the geographical distribution various than in an inland city. Therefore the measurements and hypotheses could only be checked by looking on some randomly chosen examples. Although in most cases the reviewed examples underline the made hypotheses to the impact of the tweet (especially for the distortion difference), it would be advantageous to be able to verify the interpretations of the measurements for all clustered data. Summarizing it is feasible to show the impact of many tweets on the bitly clicks, but a general model can not be computed, because of

the world's geographical irregularity.

Another interesting result of this work is, that with the distortion it was possible to determine if the bitly clicks occur local or global. The distortion measures the precision of a cluster. The higher the distortion is the more spread are the bitly clicks in the cluster and therefore the more global appear the clicks. In general, tweets make the bitly clicks more global if they are applied to all clustered data.

Remarkable are the distributions of the latitude and longitude considered separately. For the top values, where many bitly clicks are present on a certain latitude (resp. longitude), some large cities like New York, London or Tokyo could be assigned.

Besides all the geographical results, the general statistics to the data, especially to the occurrences of bitly clicks and tweets over time, returned fascinating outcomes. Events like the Patriot Day influenced the statistics obviously.

Chapter 7

Future Work

Because of the problem, that the verification of the impact of a tweet to the geographical distortion of the bitly clicks is difficult to realize, it would be interesting to generate for each coordinate (latitude and longitude) a model. For this a larger amount of data would be needed. The different models could then be compared and assembled. From the found model it would be possible to predict the geographical distribution. Nowadays prediction in social networks is an important research area. There exists already several studies concerning predicting in social media to other objectives like movies (Asur/Huberman 2010) or popularity trends of events (Gupta et al. 2012).

Another interesting future work would be to correlate the geographical distribution with other Twitter based parameters like number of followers or statuses of a user. The hypothesis could be verified, that the more followers a user has, the higher is the impact.

Finally there appeared several problems with HBase and as soon as a lot of queries are executed to HBase, some region servers shut down. It would be advantageous to look deeper into the HBase database system and especially at features like compaction of the tables to increase the performance of queries.

List of Abbreviations

API	Application Programming Interface
ARFF	Attribute-Relation File Format
CDH	Cloudera's Distribution Including Apache Hadoop
CSV	Comma-Separated Values
HDFS	Hadoop Distributed File System
JSON	Javascript Object Notation
NER	Named-Entity Recognition
PIN	Personal Identification Number
RAM	Random-Access Memory
RDBMS	Relational Database Management Systems
RT	Retweet
URL	Uniform Resource Locator
WEKA	Waikato Environment for Knowledge Analysis

Bibliography

Agarwal et al. 2011: Agarwal, Apoorv; Boyi, Xie; Vovsha, Ilia; Rambow, Owen; Passonneau, Rebecca (2011): Sentiment Analysis of Twitter Data, in: LSM '11 Proceedings of the Workshop on Languages in Social Media, pp. 30–38.

AlchemyAPI 2014: AlchemyAPI^{TM}(2014), http://www.alchemyapi.com/, accessed on: February 3, 2014.

Asur/Huberman 2010: Asur, Sitaram; Huberman, Bernardo A. (2010): Predicting the Future with Social Media, in: WI-IAT '10 Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, Vol. 1, pp. 492–499.

Bitly 2014: Bitly (2014): About bitly, https://bitly.com/pages/about, accessed on: January 6, 2014.

Cloudera 2014: Cloudera (2014): About Us, http://www.cloudera.com/content/cloudera/en/about.html, accessed on: January 30, 2014.

Do/Batzoglou 2008: Do, Chuong B.; Batzoglou, Serafim (2008): What is the expectation maximization algorithm? in: Nature Biotechnology, Vol. 26, Nr. 8, pp. 897–899. number 8

Finin et al. 2010: Finin, Tim; Murnane, Will; Karandikar, Anand; Keller, Nicholas; Martineau, Justin; Dredze, Mark (2010): Annotating Named Entities in Twitter Data with Crowdsourcing, in: CSLDAMT '10 Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk, Association for Computational Linguistics Stroudsburg, PA, USA, pp. 80-88.

Gao et al. 2012: Gao, Qi; Abel, Fabian; Houben, Geert-Jan; Yu, Yong (2012): A Comparative Study of Users' Microblogging: Behavior on Sina Weibo and Twitter, in: Masthoff, Judith et al. (Eds.): UMAP 2012, LNCS 7379, Springer-Verlag Berlin Heidelberg, pp. 88–101.

Generic Logic 2014: Generic Logic (2014): GLG Map Server, http://www.genlogic.com-/map_server.html, accessed on: February 2, 2014.

George 2011: George, Lars (2011): HBase: The Definitive Guide, O'Reilly Media, Sebastopol, CA, USA.

Gupta et al. 2012: Gupta, Manish; Gao, Jing; Zhai, Cheng Xiang; Han, Jiawei (2012): Predicting Future Popularity Trend of Events in Microblogging Platforms, in: Proceedings of the American Society for Information Science and Technology, Vol. 49, No. 1, pp. 1–10.

HBase 2014: HBase (2014): The Apache HBase^{TM}Reference Guide, http://hbase.apache.org/book/book.html, accessed on: January 29, 2014.

HeidelTime 2014: HeidelTime - a multilingual, cross-domain temporal tagger (2014): About HeidelTime, http://code.google.com/p/heideltime/, accessed on: February 3, 2014.

JSON 2014: JSON (2014): Introducing JSON, http://www.json.org/, accessed on: January 7, 2014.

Li et al. 2012: Li, Chenliang; Weng, Jianshu; He, Qi; Yao, Yuxia; Datta, Anwitaman; Sun, Aixin; Lee, Bu-Sung (2012): TwiNER: Named Entity Recognition in Targeted Twitter Stream, in: SIGIR '12 Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval, pp. 721–730.

Loss 2002: Loss, Dirk (2002): Data Mining: Klassifikations- und Clusteringverfahren, Westfälische Wilhelms-Universität Münster.

Pham/Dimov/Nguyen 2004: Pham, D. T.; Dimov, S. S.; Nguyen, C. D. (2004): Selection of K in K-means clustering, in: IMechE, Vol. 219, Part C: Mechanical Engineering Science, Cardiff, UK, pp. 103–119.

Pelleg/Moore 2000: Pelleg, Dan; Moore, Andrew (2000): X-means: Extending K-means with Efficient Estimation of the Number of Clusters, in: Proceedings of the Seventeenth International Conference on Machine Learning, San Francisco, pp. 727–734.

Ritter et al. 2011: Ritter, Alan; Clark, Sam; Etzioni, Mausam; Etzioni, Oren (2011): Named Entity Recognition in Tweets: An Experimental Study, in: Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, pp. 1524-–1534.

SMILE 2014: SMILE Text Analyzer (2014), https://smile-pos.appspot.com/, accessed on: February 3, 2014.

Stanford 2014: The Stanford Natural Language Processing Group (2014): Stanford Named Entity Recognizer (NER), http://nlp.stanford.edu/software/CRF-NER.shtml, accessed on: February 3, 2014.

Strötgen/Gertz 2010: Strötgen, Jannik; Gertz, Michael (2010): HeidelTime: High Quality Rule-based Extraction and Normalization of Temporal Expressions, in: Proceedings of the 5th International Workshop on Semantic Evaluation, ACL, Uppsala, Sweden, pp. 321–324.

Timeanddate 2014: Timeanddate (2014): Patriot Day in United States, http://-www.timeanddate.com/holidays/us/patriot-day, accessed on: February 2, 2014.

Twitter 2013: Twitter (2013): The t.co URL Wrapper, https://dev.twitter.com/-docs/tco-url-wrapper, accessed on: December 16, 2013.

Twitter Engineering Blog 2013: Twitter Engineering Blog (2013): New Tweets per second record, and how!, https://blog.twitter.com/2013/new-tweets-per-second-record-and-how, accessed on: January 7, 2014.

Weka 2014: Weka 3 (2014): Data Mining Software in Java, http://www.cs.waikato.ac.nz/ml/, accessed on: January 27, 2014.

White 2012: White, Tom (2012): Hadoop: The Definitive Guide, 3rd Edition, O'Reilly Media, 2012.

Wikipedia 2014: Wikipedia (2014): bitly, http://en.wikipedia.org/wiki/Bitly, accessed on: January 6, 2014.

Declaration

I hereby declare that I wrote this thesis on my own and followed the principles of scientific integrity.

I acknowledge that otherwise the department has, according to a decision of the Faculty Council of November 11th, 2004, the right to withdraw the title that I was conferred based on this thesis.

Spiegel, the 9th February 2014

Roger Peter Kohler