HASHTAG ADOPTION ON TWITTER
The effect of network size on hashtag malleability

Submitted in partial fulfilment of
the requirements for the award of the degree of

Master of Science
in
Artificial Intelligence

Submitted by:

Iris Monster, BSc s4061381

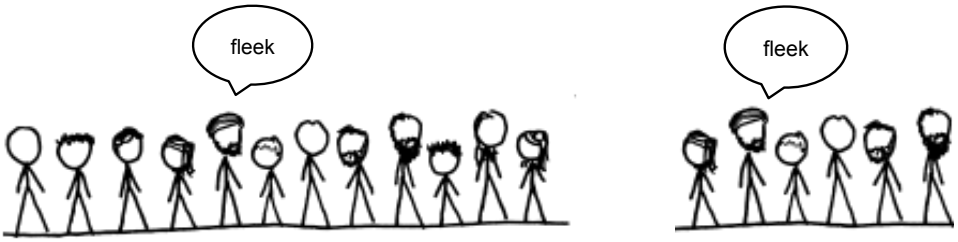
Supervisors:

Dr. Shiri Lev-Ari Max Planck Institute for Psycholinguistics
Dr. Max Hinne Donders Centre for Cognition
Prof. Dr. Boris De Ruyter Donders Centre for Cognition

Radboud University

ABSTRACT

Propagation of novel language is key in studying language change. Previous studies on linguistic representation malleability show that people with smaller networks tend to adopt changes more. This thesis studies the effect of network size on changes in language production, specifically: adoption of hashtags on Twitter. Results show that indeed network size significantly predicts likelihood of adoption. These findings contribute to understanding how rare novelties are propagated among the population and cause language to evolve.



CONTENTS

Abstract 2

Contents 3

List of Figures 5

List of Tables 6

1 Introduction 7
- 1.1 Language Change . . . 7
- 1.2 Malleable Linguistic Representations . . . 9
- 1.3 Language malleability on Twitter . . . 10
- 1.4 Competiveness of hashtags . . . 11
- 1.5 Dual role affects hashtag adoption . . . 12
- 1.6 Hashtag propagation . . . 12
- 1.7 The current study . . . 13

2 Methods 14
- 2.1 Preliminaries . . . 14
- 2.2 Data processing and storage . . . 15
- 2.3 Code to write code . . . 15
- 2.4 Exploratory phase . . . 18
- 2.5 Data harvesting . . . 21
- 2.6 Data processing . . . 29
- 2.7 Analyses . . . 33

3 Results 39
- 3.1 Results mixed model . . . 39
- 3.2 Linear regression model . . . 40

4 Discussion 41
- 4.1 Time window data analysis . . . 41
- 4.2 Collapsed data analysis . . . 41
- 4.3 Generalizability to language change . . . 42
- 4.4 Conclusion . . . 43

A Options for data collection 44

A.1 Authentication 44
A.2 Returned data structure 44
A.3 REST API 45
A.4 Harvesting lists and timelines with the REST API 46
A.5 Streaming API 46

B Description file syntax **48**

C MySQL database **50**
C.1 Java types to SQL 50

D Sample Generated Twitter Class **54**

E Unique hashtags **60**

F CSV table **62**

G Code **66**

References **67**

LIST OF FIGURES

1.1 Twitter hashtag use during 911 memorial 11

2.1 An example description file 17
2.2 Distribution of dominant language of Twitter users 27
2.3 Distribution of influencer count 27
2.4 Hashtag adoption and data points 30

A.1 A small example of a TweetArray 45

C.1 Example generated table by `TwiApiGen` 51
C.2 EER diagram of database 53

LIST OF TABLES

2.1 Highest ranked hashtag using term relevance weighting 25

E.1 Unique hashtags 61

F.1 CSV table 65

1 INTRODUCTION

Inspiration for this project comes from my supervisor Shiri Lev-Ari's findings on the influence of social network size on linguistic representation malleability (Lev-Ari, 2016). The study by Lev-Ari focuses on the malleability of internal linguistic representations, but did not test the influence of network size on language production. As linguistic changes also need to be propagated in order to speak of language change, studying what influences language production to change could provide insight in how language changes. With this Master thesis I wish to contribute to the findings of Lev-Ari's study. The research question that will be answered in this thesis is: *What is the effect social network size has on language production malleability?*

I specifically study language malleability on Twitter. In this thesis social network size is defined as the number of people someone follows on Twitter (from here on – influencers). Language malleability can be described as the degree someone copies (adopts) hashtags that his influencers tweeted. This phenomenon is studied with Twitter data, as this gives the possibility to study language malleability with real world data and not in a controlled experiment. Also, social media could contribute to language change as it gives people a lot more reach. One person invents the new term "on fleek" and it can be read all over the world.

In the following sections I will provide some background literature starting at language change in general, followed by the study by Lev-Ari. Then I will explain why I chose Twitter as the source of data and why hashtags in particular are studied. After that I will describe some previous relevant studies on hashtag adoption on Twitter. The last section of this chapter describes how these relevant previous studies come together in this current research project.

The chapter following this chapter is the Methods chapter. This second chapter contains an explanation of the steps taken to answer the research question that is asked. Chapter 2 finishes with a description of the analyses used to test the hypotheses and chapter 3 lists the results of the analyses and their interpretation. The last chapter of this thesis is the Discussion chapter. In this final chapter I will connect the results of this thesis with the literature that is described in this first chapter and what can be concluded from this study.

1.1 LANGUAGE CHANGE

When studying language change there are roughly two main components of the process: creation and propagation (Croft, 2000). Creating a new word, pronouncing something differently or coming up with another language novelty is not difficult.

Perhaps someone is not be able to pronounce the /th/ sound correctly in /teeth/ and a novel pronunciation is born. However, not every novelty leads to language change. Only when the novelty is propagated among a population we can speak of language evolution. This type of definition (that I do not speak of language change once a linguistic novelty is invented, because the novelty has to be used by a large number of people to earn the term language change) also applies to other areas where novelties arise. For example, a new *invention* is not considered an *innovation*, unless it is used by a large group of people (Baregheh, Rowley, & Sambrook, 2009). We all view Thomas Edison as the inventor of the light bulb. However, John W. Starr already patented the light bulb twenty years earlier in 1845 (Hargadon, 2003). Edison was the person who had the publicity that caused people to start using it. So although Edison did not create the first light bulb, he caused it to propagate leading to technological innovation. Ergo, propagation is at least as important as the creation.

When a new use of language or pronunciation emerges, only one person or a few people are using it. So how can something that is rare actually become adopted by a larger population? Let us assume that people assign the same weight to every piece of input they receive and learn language according to this input, i.e. they adopt a change as a function of their weighted exposure to the novel phenomenon. An invention usually starts with one person or a small group of people, so if our assumption is true novelties would never become popular, as nobody will become sufficiently exposed to it. Yet we already see language change within one lifetime. Even the British Queen does not speak the Queen's English anymore, according to a study by Harrington et al. (Harrington, Palethorpe, & Watson, 2000).

When we assume that people tend to produce language according to the norm of their input we should not see language change as novelties are always rare at first. Sapir mentioned this problem of how rare changes are propagated almost a century ago (Sapir, 1921): How can individual changes within a dialect ever become common if it is rare at first and levelled out by the norm? Sapir specifically asked how one language could evolve into distinct dialects when people would only learn from the norm. An explanation he gives is that dialects can form when two or more groups using the same language drift apart and become disconnected. This will cause the language the different groups speak to independently drift apart. This explanation by Sapir still does not explain how language drifts. If people tend to produce according to what is common in their input, the norm should wash all small changes out.

Keller refers to conformity to the norm as *the Humboldt's law* (after a nineteenth-century linguist) and states that if a speaker learns the norm of his input, after a few generations the result is "homogeneity if the starting point is heterogeneous and stasis if the starting point is homogeneous" (Keller, 1995). To rephrase Keller's words: even if the starting point is very diverse, output will converge to something

less diverse. Yet, this assumption cannot be completely true as we do see language evolve all around us.

Nettle also studied the problem coined by Sapir and refers to it as the threshold problem (Nettle, 1999). Nettle proposed three different ways for a novelty to overcome the threshold problem. The first is that some speakers are more influential than others (so their novelties are more likely to be propagated). Another explanation could be that an increase in parochialism also facilitates differentiation. When people are more parochialistic they assign more weight to input of a small group instead of that of the broader community. Take for example a small group of high school friends that come up with a new word for *cool*. The whole group will start using that word, even though no one else does. Nettle's third possible explanation is that functional biases (for example that a new linguistic change is easier to learn than the norm) also have a small influence on the likelihood of propagation.

Nettle's theories explain why some specific novelties become popular or why novelties from one person tend to find adopters more easily. However, Nettle's theories do not explain why some people adopt some phenomena more quickly than others. Here I hypothesize that the susceptibility of language adoption is a function of the social network a person is exposed to. It makes sense that someone with only two input sources assigns more weight to a new utterance from one of his sources than someone with a much larger group of input sources. This hypothesis is corroborated by the work of Lev-Ari (Lev-Ari, 2016), which is described in the next section.

1.2 MALLEABLE LINGUISTIC REPRESENTATIONS

Lev-Ari studied the degree by which participants generalize what they have learnt from one speaker to a novel speaker in relation to their social network size (Lev-Ari, 2016). Participants were asked questions about their social network[1] in order to know with how many people they regularly converse. The participants were then exposed to non-normative input in a picture selection task and tested to see if they had changed their internal threshold with a phoneme categorization task.

Lev-Ari used the distinct Voice Onset Time of /d/ and /t/ tokens for her study. A /t/ has a longer VOT, but aside from that the two phonemes resemble each other. Due to this property it is possible to manipulate the VOT of a /t/ token in such a manner that it is ambiguous whether it is a /t/ or /d/ token. During the picture selection task participants had to select pictures from sets of two pictures based on an auditory

[1] Which was specified as the number of the people the participants orally converse with in a typical week, which are not necessarily their friends.

instruction (like "The yellow toy"). Some of the trials had a noun that contained a /d/ and other trials had a noun that contained a /t/.

Participants were divided in two groups: one group that listened to manipulated /t/'s and normal /d/'s and the other group vice versa. After that they had to do a categorization task. In this task participants either listened to the same speaker or a new one. For both speakers a continuum was created with different VOTs between /dean/ and /teen/. The continuum consisted of good examples of /t/ and /d/ and several ambiguous VOTs.

The results of this study showed that social network size affects how participants generalize their shifted internal boundary between /d/ and /t/ to a new speaker when exposed to a new speaker in the categorization task. There was no significant effect of social network size on categorization when a separate analysis was ran with only the participants who listened to the same speaker. In the same speaker condition participants with small and large networks both learnt the patterns of the speaker and categorized utterances of the same speaker according to what they had just been exposed to. The fact that in the new speaker condition there is a significant effect of social network size indicates that people with smaller networks generalize the new patterns to a new speaker more. So people with smaller social networks are more likely to generalize learnt non-normative input to new speakers, which suggests that these people might also propagate linguistic innovations more. In this Master thesis research project, I study whether this phenomenon can also be observed in language production malleability on Twitter.

1.3 LANGUAGE MALLEABILITY ON TWITTER

In order to study the effect of social network size on changes in language production input and output needs to be gathered. Unfortunately, people interact with one another via many different channels, like meetings with co-workers, chatting with friends, checking social media or watching movies. People may also share information from one channel with the other. For example, after reading a news article and words used in that article can be shared wit a co-worker. Since there are so many different ways people can be influenced with regard to language use, it is not a trivial task to collect data to study language adoption.

However, when limited to only one medium-specific input and output, it is possible to minimize influence from other channels. Hashtags are an example of a medium-specific use, as it is less common to mention hashtags outside of social media. Therefore people also only receive input from hashtag usage via social media. The people someone follows on social media are his sole input of hashtags, since normal written or spoken texts usually do not contain any hashtags that are used as such. People use hashtags on almost every social media platform, like Twitter,

Instagram, Facebook et cet. Since it is difficult to combine data from different social media websites, one platform was chosen for this study. I chose to use Twitter data, because users usually allow everybody to see their posts (many Facebook accounts are private) and because people use hashtags on Twitter [2]. Another important advantage is that Twitter allows data to be harvested for research purposes.

1.4 COMPETIVENESS OF HASHTAGS

An important property of hashtags is their competiveness, i.e. the ranking (based on how often they are used) of a group of similar hashtags changes. Pontoriero and Gillie (Pontoriero & Gillie, 2012) found that for an event like the memorial of the 9/11 attack there were several competing hashtags. An early observation (when the memorial was not yet tweeted about very often) showed that the most popular hashtags were `#neverforget`, `#9/11`, `#september11th`, `#911` and `#11september`. A few hours later `#neverforget` was the most popular tag (see Figure 1.1, source: (Pontoriero & Gillie, 2012)). More interestingly, `#remember911` was the second most popular hashtag, which did not even occur in the top 5 a couple of hours earlier. People's choice of using a certain hashtag changed over time. But what influences people's decision to start using a different hashtag with almost the same meaning?

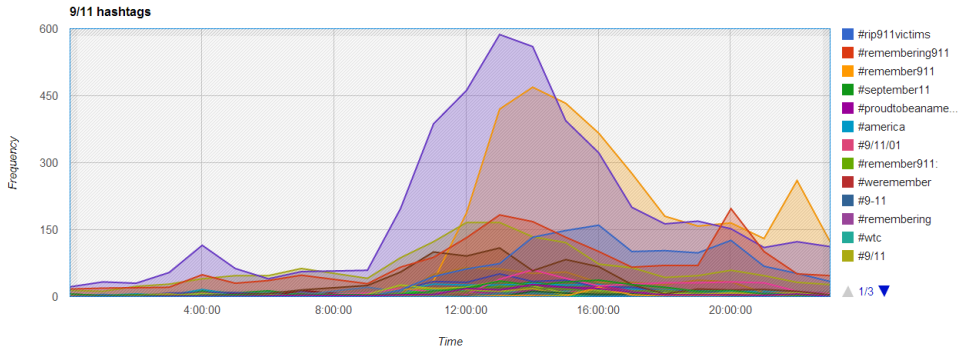


Figure 1.1: Twitter hashtag use during 911 memorial

[2] People might think that hashtags originated at Twitter, but this is not the case. Hashtags were used before Twitter on Internet Relay Chats to indicate channels. When Twitter was created hashtags were not used yet, until in 2007 Chris Messina tweeted: how do you feel about using # (pound) for groups. As in `#barcamp` Twitter began to group topics by hashtags, but found that people were skeptical towards using hashtags and did not develop the idea further. In 2009 Twitter launched the use of hashtags and people could now simply click on a hashtag to find more tweets containing the same tag. source: http://www.link-assistant.com/blog/the-history-of-hashtag/

1.5 DUAL ROLE AFFECTS HASHTAG ADOPTION

Many studies have researched hashtag adoption on Twitter. One of the more elaborate studies was conducted by Yang et al. (Yang, Sun, Zhang, & Mei, 2012). According to them hashtags serve two functions: indicating group membership and finding tweets about the same topic. They studied to what degree people are driven by these two possible reasons. Yang et al. also studied to what degree the role of the user changes his motivation to adopt a new hashtag. They collected tweets from politicians, people with a shared interest and a control group. Adoption behaviour of these three groups was analysed separately so that the results could be compared.

First Yang et al. did a regression analysis to compute per group what the regression coefficients for a large set of predictors are. Then a support vector machine was used to test the accuracy of the regression analysis on a test set of data. The results from both models showed that there were several predictors that significantly influenced adoption; popularity of a hashtag, relevance of the hashtag for that user, how many users in the user's network used the hashtag, popularity of the user, how many unique hashtags the user had used and how many characters the hashtag contained. Yang et al found that politicians were not significantly influenced by their own popularity or by the number of unique hashtags they had used.

Politicians also tend to retweet less than the shared-interest and control group. Another difference in likelihood to adopt a certain hashtag between the three groups was that the shared-interest group and control group are less likely to be influenced by the prestige of others who used that hashtag than politicians.

According to Yang et al. the influence of factors like popularity of a hashtag, how many users in the user's network use the hashtag can be attributed to a user's wish to be part of a group. The influence of relevance of the hashtag for that user shows that topic description is also a motivation to adopt a hashtag.

1.6 HASHTAG PROPAGATION

Romero, Meeder and Kleinberg also studied hashtag adoption and compared different topics with one another (Romero, Meeder, & Kleinberg, 2011). Using the Twitter API they harvested more than three billion tweets of more than 60 million users. The 500 most used hashtags were annotated by hand to divide them into eight categories: celebrity news, games, idiom (`#cantlivewithout`, `#musicmonday`), movies / TV, politics, sports and technology.

Romero, Meeder and Kleinberg measured the stickiness and persistence of hashtags. Stickiness was defined as the probability of adopting the hashtag based on one or more exposures. Persistence is defined as the degree of which repeated exposures still have significant marginal effects on adoption after the highest adoption prob-

ability. According to their results some hashtags appear to be more sticky whilst others are more persistent.

They found that, in general, topics that are politically controversial or sports-related topics show more persistence compared to idioms and music topics. Music, celebrity news and idioms are more sticky than technology, movies and sports.

1.7 THE CURRENT STUDY

Previous studies have described that it is remarkable that language changes (Keller, 1995; Sapir, 1921). If people would learn language according to what the norm prescribes, how can rare novelties can never overcome threshold problem (Nettle, 1999). Nettle has given possible explanations for language change, but these do not explain how some people tend to propagate novelties more easily than others. A possible explanation could also be that people with less input sources assign more weights to non-normative input of one of their input sources. Lev-Ari found that people with smaller networks are more likely to shift their internal representation of certain sounds (Lev-Ari, 2016).

Since social media can contribute to changes in language, it is interesting to research how for instance hashtags are propagated on Twitter. Several hypotheses have been studied, like influence of the prestige a user has on likelihood of adoption (Yang et al., 2012) or whether adoption behavior differs among different topics (Romero et al., 2011). In this thesis research project I wish to test Lev-Ari's theory on adoption behaviour on Twitter. The hypothesis is that Twitter users who follow fewer users tend to be more malleable and are more likely to adopt a hashtag compared to users who follow more people. This hypothesis could explain why some people do not adopt some hashtag even though they have been exposed to it and others do adopt that same hashtag.

METHODS

2

This chapter comprises of several parts. First the possibilities for data collection are explained, followed by a description of the exploratory phase. Subsequently the data harvesting, storing and processing pipeline are described. Lastly I specify which analyses were conducted, the results of which are listed and interpreted in the next chapter Results.

2.1 PRELIMINARIES

To avoid ambiguity and confusion I wish to explain some of the terms that will be used throughout this thesis:

Users, friends, followers and influencers

A Twitter user can have different roles. Suppose User A follows user B. In this situation, Twitter will call A a follower of B.

Definition 2.1. *(Following)*
Let A and B be users, when A follows B: $A \rightarrow B$

Since this study is about the (possible) influence one user has on another, talking about the inverse relation is useful: when $A \rightarrow B$ we also say that B *influences* A. So B is called an *influencer* of A.

When viewing an account on Twitter the followers are labelled as 'followers', however the influencers are listed under the tab 'following'[1].

Posts

On Twitter, people post tweets. Sometimes they are referred to as a status, a post or a tweet, which are considered synonyms. A tweet can be one of these things:

- *A regular tweet*
  Here the tweet is original text from the person who posted the tweet.
- *A retweet*
  When person A posts something, person B can retweet the tweet. This means that nothing is added to the original text and the post is now also visible on

[1] In the Twitter API documentation, B is called a *friend* of A. I will not use this term, since it suggests a symmetrical relation (that A and B follow each other)

the timeline of person B. This means that followers of person B now also see the tweet (they can see that it is a retweet).

- *A quote*
  This resembles the retweet, but person B has now also added something to the original tweet.

- *A reply*
  People can reply to a tweet from another user by pressing the reply button. The user who is being replied to will be mentioned in the tweet automatically. Replies are shown under the 'Mentions' tab and if the user who is replied to follows the person who replied, it will also be seen on his homeline. Other people that follow both users will also see the reply.

Unless a particular category is specified, a 'tweet' may refer to any of these categories.

2.2 DATA PROCESSING AND STORAGE

Both the REST API and Streaming API yield `JSON` objects that need to be processed and stored. In order to manipulate `JSON` objects in `Java`, the `JSON` String representation should be parsed. There are multiple open-source `JSON` libraries to perform this task. However these libraries generate 'untyped' `Java` objects.

Apart from parsing and representing `JSON` objects as typed `Java` Classes it would be nice to have methods to store (and read in) these `Java` objects in a MySQL database.

To solve these problems I created a program that generates `Java` code that can parse and manipulate Twitter objects. The program would also generate the structure of a MySQL database and the generated `Java` code contained methods that could store objects in the database. Unfortunately, in the final setup the database was not used, because adding large quantities of data to the database slowed down data collection too much. The data was stored in compressed `JSON` files (json.gz). Information about how the database was generated can be found in Appendix C.

2.3 CODE TO WRITE CODE

2.3.1 *TwiApiGen version 1*

While handcrafting a `Java` program for parsing the Twitter User object I realized that a large amount of similar work has to be done for all other Twitter objects. So the idea arose to create a `Java` program that writes a `Java` program that parses the `JSON` objects that Twitter returns. This program is called `TwiApiGen` and it was programmed to generate code to parse the Twitter `JSON` objects and offers access

routines to process the parsed data. Using TwiApiGen has a number of advantages compared to hand crafted Java code. First of all: creating the code by hand would have taken a lot of time and was prone to mistakes, due to the size and complexity of the JSON data objects. Secondly, quite often Twitter adds or changes something to the data that it returns. Whenever this happens, new code can easily be generated. Code used for this project will be uploaded to GitLab, see Appendix G.

2.3.2 *Twitter essentials: Description files*

TwiApiGen uses description files to know the structure of the JSON objects it can expect. These description files are easy to write, since they only capture the essential information needed to generate the Java Code. The syntax of the description file looks like a type (structure) definition. To parse a description file I used a tool called ANTLR (Parr, 2013). Instead of writing Java code to parse description files myself, I wrote a grammar that defines the description file language (see Appendix B) and let ANTRL generate a parser for it.

The description file (see Figure 2.1) contains a list of all fields the object can have [2]. The type of the field can be any Java native data type (like a String or Integer), a new object type (like Tweet or User) or a list of any of these data types (a list of Tweets, or a list of integers). The names of the fields or classes mostly correspond to the names Twitter gives these fields, but sometimes I had to name a structure myself. The comments from the description file (which are copied from the Twitter API documentation) will be automatically transferred to Java code as JavaDoc (for example to add comments to the getters and setters). This way the Twitter API documentation is directly available for programmers using the API.

The example description file (Figure 2.1) shows that the style resembles Java code; the comments are indicated in Java style, the fields are structured like Java fields (`type name;`) and arrays are indicated by two square brackets preceding the variable name.

2.3.3 *Types in TwiApiGen*

When the type of a field is a Java native object like String, Long, Boolean or Integer, TwiApiGen simply uses the Java native class as the type indicator for that field. In all other cases TwiApiGen creates a custom class.

Although Java has data structures for arrays, it was not possible to directly use these as the type indicator for an array. In the example of UserIDArray there is a Long[] called ids. Ideally, a field Long[] ids would be generated in the generated Java class for UserIDArray. That is not possible, because custom methods need to

[2] Note that not all fields are always present for every object returned by Twitter.

```
UserIDArray
{
  /*
  ** Id of the previous cursor
  */
  Long previous_cursor;
  /*
  ** An array of user IDs
  */
  Long [] ids;
  /*
  ** String representation of the previous cursor
  */
  String previous_cursor_str;
  /*
  ** Id of the previous cursor
  */
  Long next_cursor;
  /*
  ** String representation of the previous cursor
  */
  String next_cursor_str;
}
```

Figure 2.1: An example description file

be added to the object class and these can only be placed in a custom class. For all arrays `TwiApiGen` created a class named after the type that the array holds (in this example a `LongArray` class). This class does contain a field with the `Java` native array type.

There were also a lot of new classes that were generated specifically for Twitter data. There is a class called `Tweet`, but there are also cases where an object contains an object of a generated class. Every tweet data object for example contains a `Hashtag` array that contains `Hashtag` objects (or is empty). A `Hashtag` contains the text of the hashtag (the #-sign is removed) and a list of indices to indicate where in the 140 characters long tweet the hashtag was. `TwiApiGen` generates a class called `HashtagArray` that contains a list of `Hashtag` instances.

2.3.4 *Generated methods and name giving*

Every generated class contains methods that parse the `String` representation of the `JSON` object (which is what Twitter returns) to `Java` objects. `TwiApiGen` also generates a method that converts the `Java` object back to a `JSON` object (the original `JSON` object is not kept in working memory as that would require too much memory space). The conversions from or to `JSON` are done with the help of a package called `JSON` Simple (Fang, 2016). `TwiApiGen` creates methods for getting or setting a certain field (like `getPreviousCursor()`).

Methods are generated in a hierarchical structure. When a `JSON` string is read and the parse method is called of appropriate data structure (for example Tweet.parse()), objects that are nested within that structure are also created automatically. In other words, the parse method of Tweet calls the parse methods of the objects that are nested within the Tweet object. This hierarchy also applies to the access of information. For example, a `Tweet` contains a `User` object and that contains the `username` of the user, the `username` can be accessed with `tweet.getUser().getScreenName()`.

The names that are written in the description file correspond roughly with the names that Twitter gives for these `JSON` fields. The shape of these names is not always appropriate for generated `Java` code. In order to comply with the standard `Java` coding conventions, my `Java` code uses lowerCamelCase format for identifiers (field and method names). Again, conforming the `Java` conventions, generated class names are written in UpperCamelCase format.

2.4 EXPLORATORY PHASE

Before harvesting the data needed for our study, the following questions needed to be answered:

1. Is it possible to collect all tweets from specific users?
2. Which Twitter-enforced limitations constrain harvesting?
3. Which time/space limitations constrain harvesting?
4. What are the requirements for an appropriate topic for our study?
5. Is it possible to find a list of 'competing' hashtags given such a topic?

In order to answer these questions, I conducted an exploratory data study, which led to the following conclusions:

ad 1. There are three methods we can use to collect tweets from a user: using the REST API with a query, using the Streaming API and using the web interface

https://twitter.com/. A more elaborate explanation of the REST API and Streaming API can be found in Appendix A

ad 2. The different harvesting methods have different limitations. The REST API has bandwidth limitations (fixed number of requests per time interval). For example, it is only capable of returning the last 3200 tweets of a user. The Streaming API can be used to harvest tweets, but it only yields a fraction of all tweets and harvesting must be done in real time (cannot be done afterwards). Finally the web interface can be used to harvest queries. Although there seems to be no limit to the number of tweets that can be gathered this way, Twitter put in a lot of effort to prevent automatic harvesting.

ad 3. There are no real raw space limitations for current hardware. Storing the Streaming API result for one single day costs about 2GB of data (compressed). While gathering, CPU performance is not a real issue as network latency and Twitter time restrictions are significantly dominant.

ad 4. A good topic for this study should:

- Be trending (or popular) enough to generate enough Twitter traffic.
- Be specific enough to be distinguishable from other events.
- Have different competing hashtags, with more or less the same meaning in order to be enable to detection of hashtag adoption.

ad 5. It is possible to obtain a list of competing hashtags using only the REST API. Using the `GET statuses/lookup` method it is possible to request tweets with a search query. Tweets that are returned often also contain other relevant hashtags that can be filtered out manually. Unfortunately, just using the REST API is not very reliable as tweets that do not satisfy the search query could contain relevant hashtags that are now never found.

2.4.1 *Initial pipeline*

At first I used the REST API to get more familiar with the data and I worked out a possible pipeline for harvesting the data if I would only use the REST API. The pipeline was as follows:

- Choose a keyword or set of keywords
- Request recent posts with these keywords using the `GET statuses/lookup` method.

- Search these recent posts for tweets that contain hashtags. Because these hashtags were used with one of our keywords, the hashtag probably has something to do with this topic. Manually create a list of relevant hashtags.
- Select users who used one or more relevant hashtags in the recent posts.
- Harvest full tweet history of these selected users with the `GET statuses/user_timeline` method.
- Request influencer id's from the users with the `GET friends/ids` method.
- Harvest full tweet history of the influencers using the user ID's.

During the exploratory phase tweets mentioning the campaign of Bernie Sanders (who was at the time running for president in the United States of America) were harvested. This topic had a few competing hashtags and appeared usable. However, once the requirements for a good dataset were clear, this topic could not be used. The three requirements are explained more elaborately below.

First of all, selecting different keywords to start with could lead to different hashtags and therefore different users. It is possible that some hashtags are not used along one of the keywords and are therefore never found. It would be best to find the most complete possible list of hashtags about a topic, but this cannot be assured when manually selecting keywords.

Secondly, Twitter's documentation is unclear regarding which tweets are returned when using the `GET statuses/lookup` method. It is likely that Twitter uses an algorithm to compute how relevant a tweet is. Obviously, a random sample is preferred. The two issues above were solved by using data from the Streaming API. The Streaming API does return a random sample of tweets and a term relevance weighting based algorithm was used to find hashtags that are competing for a certain topic. More details on this algorithm can be found in Section 2.5.2.

Lastly, at the time data about Bernie Sanders was harvested. people were tweeting about him for about half a year (since he announced his candidacy in April 2015). Most users had already converged to using a particular hashtag, unless something happened that caused the invention of a few new hashtags. It would be best to start tracking adoption behaviour right from the start, especially since Twitter allows us only to retrieve the last 3200 tweets. Therefore, I sampled a few users from a large group of users who tweeted about Bernie Sanders in the first week of November 2015. This sample was drawn according to the distribution of influencer counts. Tweets of all influencers of this sample of users were harvested and the date of the oldest tweet Twitter would return was checked. Almost every user followed a few influencers, for whom it was not possible to go back further than a few weeks. Since

it would be best to obtain tweets from all influencers, it was not possible to study adoption since people started tweeting about Sanders' candidacy.

A good topic would be concerning something that happens without knowledge beforehand. Movie premiers cannot be used, because they are long anticipated and people will have already tweeted about the movie before the actual release. The other criterion is that people use different hashtags that have the same meaning. For example, organized events usually promote one single hashtag for people to use so they can monitor what has been tweeted about the event and are also known beforehand. The third criterion is that it is best that the meaning of the hastags is more or less the same, because even though people are tweeting about the same topic, it could be that there are two sides (like in a political debate). It could happen that a user does not adopt a hashtag and uses another, simply because the first does not reflect his opinion.

Once the system for data harvesting was finished I started searching for a new topic again.

2.5 DATA HARVESTING

The data-harvesting phase consists of the following steps:

1. Find an appropriate topic (the terrorist attack in Brussels in March 2016)
2. Find all relevant hashtags for that topic (by using a tf-idf based algorithm).
3. Find users who used at least one of these hashtags.

Using the Streaming API we found $76,211$ users who used any of a list of 177 hashtags that were marked 'relevant' for this topic. Due to the rate limits it was not possible to harvest tweets for these $76,211$ users and all of their influencers. Therefore I first filtered based on use of the same language (Dutch) and then I took a random sample that was feasible to harvest. This left around 979 users. In the following subsections I will explain the harvesting steps in more detail.

2.5.1 *Finding a topic*

As described in section 2 of this chapter not every topic can be used for this study. In order to collect all possible instances of hashtag adoption for a particular topic, tweets must be collected starting at the first usage of a relevant hashtag. Furthermore people have to use several competing hashtags. Tweets regarding the tragic terrorist attack that happened in Brussels on March 22 2016 did fit these criteria. I was able

to start harvesting data right after it happened and people used many different, competitive. Some example hashtags include: `#brussels`, `#prayforbelgium`, `#zaventem`, `#jesuisbruxelles`. For a full list of hashtag see Appendix E.

2.5.2 *Finding all relevant hashtags*

At the time a server was running the Streaming API which gathered a sample of tweets from around the world. This was our first source of data. I developed two algorithms for finding relevant hashtags, both based on the notion of term relevance weighting (Robertson & Jones, 1976). The first algorithm uses co-occurrence to define relevance; the second algorithm uses timing to define relevance. These two approaches were chosen as The output of both algorithms was used to hand-pick a set of relevant hashtags. But first, lemmas had to be retrieved from the tweet texts to use for the term relevance weighting.

Lemma extraction

In order to get closer to the meaning of words (compared to their word form) I annotated tweets with the Stanford parser (Klein & Manning, 2003; Manning et al., 2014). This way word lemmas were contained, which are better suited for the frequency counts that were used. Only the English parser was used on tweets that were labelled 'English' by Twitter. It would have been better to use different parsers for more languages, but time restricted this possibility. The decision to only use Dutch users [3] was made later, so in hindsight it would also have been better to use a Dutch parser. Many tweets were labelled 'English', therefore using the English parser seemed the best option.

Parsing the gigantic amount of tweets with the Stanford parser turned out to be a challenge, but the process was accelerated by parsing in parallel. I created a Server/Client model using `Java` RMI (Remote Method Invocation), which was able to divide the work for this task to 20 computers/CPUs at the same time. The resulting speed-up was almost linear: 15 million tweets were analysed in 8 hours which otherwise would have cost almost a week.

To collect all lemmas a program was created that reads the stored tweets from the Streaming API and searches for tweets that were labelled "English" by Twitter. This program is called the Stanford Client. The Stanford Client then sends the tweet text to the Stanford Server. The Stanford Server keeps track of a list of connected remote Stanford Workers that do the actual parsing. The Server sends the text to

[3] Users that, according to a frequency count of the language-labels Twitter gives tweets, mostly tweet in Dutch. Although these users might not live in The Netherlands, for abbreviations purposes these users will now be referred to as 'Dutch' users.

a Worker that is available, i.e. did not have a full waiting queue. Once the Worker parsed the text, it sends the information back to the Server that in its turn sends the parsed information back to the Client.

Relevance based algorithm, co-occurrence

In this version of the algorithm, tweets are considered relevant if the lemma 'brussels' occurs in the tweet, otherwise the tweet is considered non-relevant. Lemmas used in relevant tweets are counted as relevant (note I only count lemmas once per Tweet). If a lemma is used in a non-relevant tweet, the occurrence is labelled as 'non-relevant'.

Relevance based algorithm, time-based

When a lemma occurs in a tweet written before the actual time of the event (between Mon Mar 21 08:00:00 CET 2016 and Tue Mar 22 08:00:00 CET 2016) its use is counted as 'non-relevant'. When a lemma occurs in a tweet after the time of the event (between Tue Mar 22 08:00:00 CET 2016 and Wed Mar 23 08:00:00 CET 2016) its use is labelled 'relevant'. Again, lemmas are counted once per tweet.

Term relevance weighting

For both algorithms I calculated a lemma weighting score, based on traditional term relevance weighting (Yu & Salton, 1976; Robertson & Jones, 1976). Consider for each lemma the following values:

r	-	the number of times a lemma is used in a relevant tweet
n	-	the number of times a lemma is used
R	-	the number of relevant tweets (independent of the lemma)
N	-	the total number of tweets (independent of the lemma)

The goal is to find a formula that scores a lemma. A high scores suggests a good (relevant) lemma and a low score should suggest a less good lemma.

$$\text{score} = \frac{r}{R} \tag{2.1}$$

The score formula 2.1 calculates the fraction of occurrences of the lemma in relevant tweets. For good lemmas this value is close to 1, for bad lemmas this value is close to 0.

$$\text{score} = \frac{r}{R - r} \tag{2.2}$$

Formula 2.2 shows the same behaviour, but its effect is amplified a little.

Only the ratio of the lemma's occurence in relevant tweets is not enough: for example the lemma 'the' occurs in many relevant tweets, but is not really a relevant lemma since it occurs in a lot of non-relevant tweets as well. Realize that:

n-r - the number of times a lemma is used in a non-relevant tweet
N-R - the number of non-relevant tweets

$$\frac{n-r}{N-R} \tag{2.3}$$

Formula 2.3 describes the fraction of occurrences of the lemma in *non*-relevant tweets. For good lemmas this value is close to 0, for bad lemmas this value is close to 1.

$$\frac{n-r}{N-R-(n-r)} = \frac{n-r}{N-R-n+r)} \tag{2.4}$$

Like explained before, the formula 2.4 shows the same behaviour, but its effect is amplified. Combining formula 2.2 with 2.4 leads to:

$$\text{score} = \frac{\frac{r}{R-r}}{\frac{n-r}{N-R-n+r}}$$

Usually a log-transform is used when comparing relevance between lemmas (this is not needed when only ranking lemmas). To prevent division by zero the ranking is smoothed, resulting in the final scoring function:

$$\text{score} = log\left(\frac{\frac{r}{R-r+0.5}}{\frac{n-r}{N-R-n+r+0.5}}\right)$$

The co-occurrence algorithm labels hashtags 'relevant' that occur in tweets that contain the lemma 'brussels'. Probably all hashtags collected this way are relevant for this topic. However, it is possible that there are hashtags and keywords that never co-occurred with the lemma brussels. For example, when people from another language tweet about Brussels they may use another name (like Bruselas in Spanish). This is why I implemented the second algorithm.

Results

Table 2.1 shows the ten hashtags that had the highest ranking according to the algorithms. I manually went over the two lists of weight values for both the time-based algorithm as related based algorithm to select hashtags that were concerning

	Co-occurence		Time Based	
Rank	Hashtag	Score	Hashtag	Score
1	#zaventem	9.43	#ebgodgavemeyou	6.98
2	#belgium	9.31	#brusselsattacks	6.92
3	#breaking	9.24	#brusselsattack	6.91
4	#brusselsattack	9.03	#prayfortheworld	6.89
5	#bruxelles	8.98	#prayforbrussels	6.88
6	#prayfortheworld	8.97	#prayforbelgium	6.88
7	#maalbeek	8.86	#zaventem	6.87
8	#maelbeek	8.82	#jesuisbruxelles	6.86
9	#isis	8.78	#brussels	6.84
10	#prayforbelgium	8.77	#brusselsairport	6.84

Table 2.1: Highest ranked hashtag using term relevance weighting

the Brussels attacks. It was not possible to select an automatic threshold (for example the #ebgodgavemeyou or #breaking are not considered a competitive hashtag). The co-occurrence algorithm yielded 95 relevant hashtags and the time-based one 131. There were duplicate hashtags in these two lists, so after these were removed I had obtained a list of 177 relevant hashtags.

2.5.3 *User selection*

The next step is to find users who used one or more of these hashtags. I searched seven days of Streaming data for users who tweeted or retweeted with a relevant hashtag and made a list of their userids. The resulting list contained 76, 211 unique users and I harvested their tweets and network information (the userids of the influencers they follow). I did not intend to harvest their full tweet history of the latest 3200 posts, but only tweets posted after the Brussels attacks happened. Since I cannot tell the GET statuses/user_timeline method to only give me tweets since a given timestamp, I simply asked for 200 tweets in every request. If the last tweet in that tweet list had a timestamp of before March 22 2016, I could stop harvesting for that user. The full tweet objects (including the last array that contains at least one tweet posted before the attacks) were stored in a compressed JSON file.

As I was harvesting tweets for this amount of people I realized that, even though I often did not need to request 3200 tweets, it took a lot of time to harvest tweets due to the restrictions of the REST API. The harvesting program could harvest tweets of 21 users per minute on average, so I had to estimate how much time it would

take to harvest the tweets of all the influencers: The sum of the influencer counts of the $76,211$ users is $101,455,590$. Let us assume there are some duplicate influencer user IDs in this list and that removing duplicates reduces the amount of influencers by twenty percent. This would mean $81,164,472$ unique influencers whose tweet history needs to be harvested. Due to the rate limits of the REST API retrieving the tweet history of that many people would take more than seven years, even with six authentication keys.

Needless to say this is not feasible, so a sample was taken from the original $76,211$ users based on language, which reduced the number of users to 2575. To reduce the harvesting time even further a distributed sample was taken of these 2575 users, which reduced the number of users to 979.

Language filtering

With the first filter I wanted to increase our chances of finding users with overlapping networks (i.e. share influencers). This would decrease the amount of influencers to harvest, but would not lead to less users to study. I thought that a summary of the language and geo location information of the users would provide some helpful insights.

To create such a summary I took the harvested tweets of the $76,211$ users. For every user I computed a frequency count of the language field in his tweets. I left out retweets, because retweets say less about the tongue of the user than his own tweets. For every user I selected the language that had the highest frequency and called it the user's dominant language. I also collected geo location when present in the tweet, but this field was empty far more often than filled, so could not be used. A user object also contains a language field, but this remains the same and Twitter does not document what exactly is described with this user-language field, so I preferred counting language frequencies of the tweets.

The figure 2.2 shows a histogram of the ten languages that were most often the dominant language of a user.

I was looking for a group of users whose tweets I could understand, that probably live close to Brussels and who probably have overlapping networks. I chose to take the 2575 users whose dominant language in recent tweets is Dutch. I assumed that, when filtering for a specific group of users based on their dominant language, these users tend to partly follow the same influencers.

Distributed sample

Unfortunately, harvesting the unique influencers of all 2575 Dutch users would still take too long. Some of the Dutch users have extremely large networks. For example,

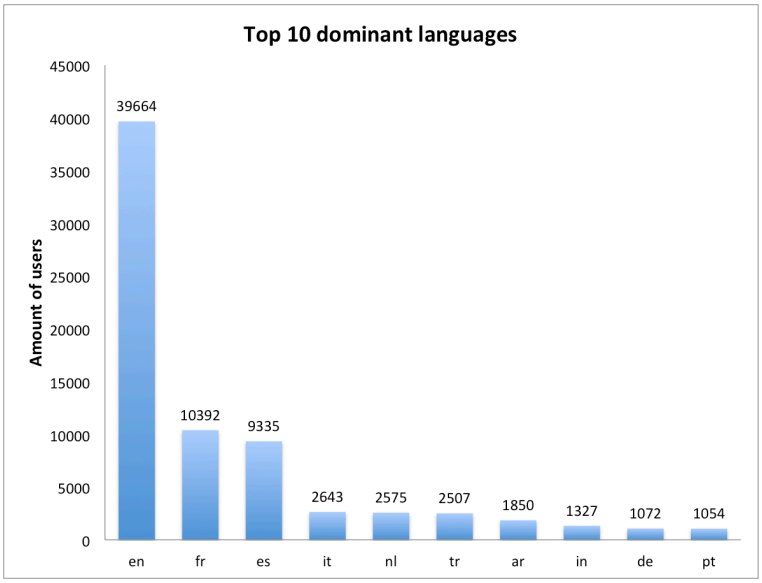


Figure 2.2: Distribution of dominant language of Twitter users

the user with the most influencers has $40,327$ influencers. Harvesting the tweets of these influencers would take almost 4 days. Figure 2.3 displays the influencer counts of the 2575 Dutch users.

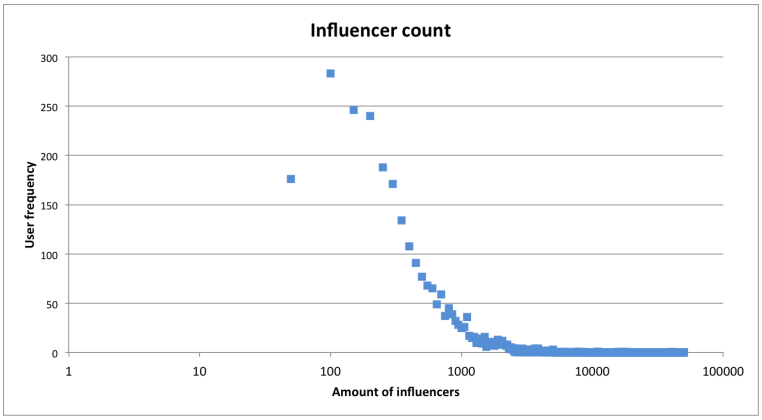


Figure 2.3: Distribution of influencer count

The available time for harvesting influencer tweets was a week. The amount of time it will take to harvest influencer data depends on how many users I wish to study and how many unique influencers they have. Because I wanted to try a few parameters, I created a `Java` program that would draw a distributed sample and compute how long it would take to harvest tweets of influencers given:

- *A cut off value*
  Only users with an influencer count below this value will be picked.
- *A bin size*
  This value indicates how large every bin is. A bin size of 50 means that users with an influencer count of $0 - 50$ (inclusive) will be put in bin 1.
- *A reduce-to factor*
  This factor is a value between 0 and 1 and indicates the proportion of users I want to have.
- *The connections*
  I had already harvested the influencer lists of the $76,211$ users, so for this program I made a folder with just the influencer lists of the 2575 Dutch users.

The sampling program would first cut off users who have an influencer count that is above the cut off value. Subsequently the bins are filled. The program computes the amount of users in the bin and multiplies the amount with the reduce-to factor. This result is the amount of random draws the program takes of that bin. Once the program is done drawing users from every bin it computes how many unique influencers these users have combined and shows how long it would take to harvest tweets for this amount of influencers.

I decided on a cut off value of 3000, because below this threshold the difference between two subsequent users (sorted by influencer count) is always less than 100 and above it some gaps are larger than 100. The sudden gaps between two consecutive users indicate that users with more than 3000 influencers are outliers. This initial cut off value reduces the amount of Dutch users from 2575 to 2506 users. This reduction seems useless, but as this cut cuts out users with high numbers of influencers, harvesting time is reduced by twelve days.

Then I tried several reduce factors. A reduction to 0.4 of 2506 led to 979 Dutch users who have $229,926$ unique influencers, which would take a little more than 7 days to harvest. This duration of harvesting was acceptable and I harvested the tweets of these influencers.

2.5.4 *The resulting data*

After I harvested the tweet history of the influencers of 979 Dutch users I finished the harvesting phase. The following summation describes the raw data:

- A list of 177 hashtags that are related to the terrorist attacks in Brussels.
- A folder with 979 files that contain tweets of 979 Dutch users. These files contain the full Tweet objects as returned by the REST API and are compressed to `json.gzip` files.
- A folder with 979 files that contain a list of influencer user ID's for 979 Dutch users.
- A folder with $229,926$ files that contain tweets of the $229,926$ influencers.

This data is called raw, because it cannot directly be used to test our hypothesis. The raw data first needs to be processed, as described in the next section.

2.6 DATA PROCESSING

The raw data, described in the previous section needs to be processed so it can be analysed. I want to analyse the behaviour a user exhibits after he or she has been exposed to one of the relevant hashtags by one of his influencers (exposure 1) and before one of his influencers uses the same hashtag again (exposure 2). Working with windows between exposure 1 and 2 allows us to control for certain factors, like the activity of the user within that window and hashtag use of the user before an exposure.

The final data ought to be a CSV file with a row for every hashtag a user could have been influenced by, which will now be referred to as an influence data point. (footnote: For now I assume that every user always saw every tweet posted by his influencers. Of course, this is very unlikely. Later I filter the data for windows where the user used any relevant hashtag. I do keep the exposure counts as I assume that although the user probably has not seen all exposures, but the count will still be a good relative frequency count (i.e. presumably the higher the exposure rate, the higher the actual exposure)).

An influence data point is defined as any of the 177 relevant hashtags used within any type of tweet (retweet, tweet, quote or reply) by one of the 229926 influencers at any time between March 22 2016 and April 5 2016. The influence data point can be any type of post. Every follower of an influencer can see retweets, quotes and original tweets. With replies it is more difficult to verify if the user could have seen the reply. To simplify the problem, it was assumed that users could see the replies. It

is possible that therefore hashtags were erroneously counted as exposure. The range of 2 weeks was chosen to ensure a wide enough range of potential influence.

An influence data point is matched to all users who follow the influencer who posted the hashtag. This means that there can be multiple rows with the same influence data point as it happens that multiple users follow the same influencer. The algorithm I wrote to create the data for analyses would then search for the next influence data point with the same hashtag tweeted by any influencer of the user. Once the time window of two consecutive exposures to the same hashtag was constructed, the program would summarize the behaviour of the user before the first influence and the behaviour within the time window. The collection of knowledge of influence data point 1, of a user, of influence data point 2 and of the behaviour of the user within and before the window is called a final data point and is exactly one row in the CSV file for analyses.

See Figure 2.4 for an example timeline. The user is influenced by 4 different influencers, of which 3 exposed him to the same hashtag(`#brussels`). With this

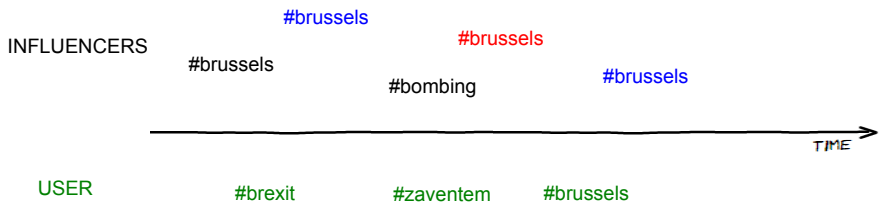


Figure 2.4: Hashtag adoption and data points

timeline the program would create 5 final data points (note that there are overlapping time windows):

1. Start of the time window is the black `#brussels`
   End of the time window is the first blue `#brussels`
   The user's behaviour within this time window is *tweeted about something else*: `#brexit`.

2. Start of the time window is the first blue `#brussels`
   End of the time window is the red `#brussels`
   The user's behaviour within this time window is *used another relevant hashtag:* `#zaventem`.

3. Start of the time window is the black `#bombing`
   There is no next exposure to that same hashtag, so the end of the time window

is set to April 5 2016.
The user's behaviour within this time window is *used another relevant hashtag*: `#zaventem` and `#brussels`.

4. Start of the time window is the red `#brussels`
   End of the time window is the second blue `#brussels`
   The user's behaviour within this time window is *adopted*.

5. Start of the time window is the second blue `#brussels`
   There is no next exposure to that same hashtag, so the end of the time window is set to April 5 2016.
   The user's behaviour within this time window is *inactive*.

2.6.1 *Data processing algorithm*

Here I describe exactly how the program for processing the raw data works. The program first reads the data and filters for usable data. Then it combines the components together and computes summaries.

Filtering the raw data

The raw data contained a lot of tweets that are completely unrelated to the Brussels attacks and the original elaborate `JSON` files also had to be compressed. First I read all of the tweets and user connections in and created instances of the following `Java` Classes:

- *User data points*
  Every tweet a user posted between March 22 2016 and April 12 2016 was stored as either a:
  - *User relevant data point*
    This object is created whenever the user tweeted (not retweeted) one of the relevant hashtags. (Footnote: Hashtags mentioned in a retweet of a user are not taken into account as it is unclear if the user truly meant to use that hashtag or if he retweeted a tweet for another reason.). If a tweet contains multiple relevant hashtags, multiple instances of this Class are created. This data point also contains additional information like a timestamp, text of the tweet et cetera.
  - *User non-relevant data point*
    This object holds any type of tweet that did not contain any of the relevant hashtags. These data points were useful to decide if a user tweeted something within a certain time window.

- *Influence data points*
  A hashtag tweeted (any type of tweet) by an influencer. If a tweet contains multiple hashtags, multiple instances were created.

- *Connections*
  Per user I created a list containing the user ID's of his or her influencers.

Reading in the data and creating these datasets took about six hours. Because I often ran the program that created the final data, I stored the data described above as serialized `Java` objects. This reduced reading time to a few minutes.

Combining influence to user

To combine the influence data points to a user and construct the final data point I used the following (pseudo)algorithm:

```
For every influence data point x
 with hashtag h posted by influencer i at time t_0:
  For every user u who follows i:
   Create a final data point f
   Find first influence data point y with h,
    at time t_1 larger than t_0 posted by any of u
   If y does not exist, set t_1 to April 5 2016
   Get user data point before t_0 and
    summarize behaviour
   Get user data points between t_0 and t_1 and
    summarize behaviour.
   Add data point information and summaries to f
   Print f to CSV file
```

A full description of the columns of the data can be found in the Appendix F. Combining the data led to $287,799$ final data points.

Collapsed data

The just described data file is created to analyse adoption behaviour of a user between two consecutive exposures, but it could not be used in this format to analyse if users with smaller networks are more likely to use more, different hashtags. In order to do such test that hypothesis an additional data file was created. The resulting data file contains a row for every user who used any relevant hashtag in a time window and describes:

- The number of unique hashtags the user has been exposed to.

- The number of times the user was exposed to a relevant hashtag.
- The number of hashtags the user used.
- The number of hashtags the user used *after* one of his influencers used that hashtag.
- The number of influencers the user has.

The resulting data will be referred to as the collapsed data.

2.7 ANALYSES

This section describes what the two models that were used for analyses and which predictors were used.

2.7.1 *System specifications*

The analyses listed were run on a Windows computer with R version 2.9.2, package LanguageR version 0.955. The method used for the mixed models is called `lmer()` and the method used for the linear regression analysis is `lm()`.

2.7.2 *Mixed models*

Jiang explains how a mixed model works by first explaining how a linear regression model works (Jiang, 2007). A mixed model resembles a linear regression model. A linear regression model is expressed as $y = X\beta + \epsilon$, where y is a vector of observations, X is a vector of known predictors, β is a vector of unknown regression coefficients and ϵ a vector of errors. When a linear regression model is fit, the regression coefficients and errors are estimated based on the observations and predictor values. In a linear regression model the regression coefficients are considered to be fixed. However, sometimes it is preferred to consider these coefficients random, e.g. when observations are correlated. This is the case when, for example, the data consists of multiple observations of the same participant. In this example it is preferable to add a random effect (that is not observable) that may be different for every participant. In a mixed model these random effects can be added. A linear mixed model can be expressed as $y = X\beta + Z\alpha + \epsilon$. The difference between this expression and the linear regression model is obviously the term $Z\alpha$ which denotes the matrix Z of observed values and a vector of random effects α.

Adding slopes to a random effect to indicate that the predictor (the slope) has a special interaction with the random effect (Bates, Mächler, Bolker, & Walker, 2014). This interaction means that the predictor might have different effects for every

level of the random effect, e.g. a different effect for every participant. Slopes can have multiple purposes: it can be used to obtain shrinkage estimates of regression coefficients (Efron & Morris, 1977) or account for lack of independence in the residuals due to block structure or repeated measurements (Laird & Ware, 1982).

Selection of predictors

A linear mixed effect model was used to analyse the window-based data. The aim was to test which factors influence likelihood of adoption. The dependent measure is therefore the variable 'adopted', which is a factor that equals 1 if the user has used the hashtag he is influenced with before being exposed to the same hashtag again (i.e. within the time window of two exposures). Of course, one of the predictors is the social network size of the user, because this is the main research question of this study. Social network size is defined by the number of influencers the user has and is named `userInfluencerCount`. Several other factors were added to the analysis as well, because beforehand I hypothesized that these factors could also influence the likelihood of adoption. It is best to add these other factors to the analysis as well, instead of only using social network size as a predictor. It could be that social network size influences another factor and that factor influences likelihood of adoption. For example, the number of influencers a user has, probably influences the number of unique influencers who have used the hashtag. It could be that not social network size predicts likelihood of adoption, but actually the number of unique influencers who used the hashtag does. Without this extra factor, the effect is erroneously attributed to social network size.

The number of influencers (named `uniqueInfluencers` in the CSV file) who have exposed the user to this hashtag is used as a predictor as it makes sense that the more influencers in a user's network the more likely the user is to adopt the hashtag. The factor is also added, because (as mentioned above) it could be that social network size influences this factor and I do not want to attribute influence to the wrong factor.

The language of the influence tweet is taken into account. The column named `influencerLanguageIsDominantLanguageUser` equals 1 when the user's dominant language (between March 22 and April 12) equals the language-label Twitter assigned to the text of the influence tweet. My hypothesis is that an exposure in the same language as the user's dominant language will be more likely to cause the user to adopt the hashtag.

Two other important factors that were added describe how dominant this hashtag was (until and including the first exposure of the time window) in the user's input and in the user's output. The factor `isDominantHashtag` equals 1 if this hashtag was the hashtag the user saw most often. This factor is added to the model, because it makes sense that a hashtag that the user saw the most is more likely to be adopted.

The factor `hashtagDominantPastUse` describes the user's own hashtag usage. This variable equals 1 if the user used this hashtag most often before the time window, 0 if the user had not used any hashtag yet and -1 if the user used another hashtag most often before this time window. My hypothesis concerning this factor is that a user who has not used any hashtag is more likely to adopt the hashtag than when he was already using another hashtag. A user who was already using the hashtag will probably use it again.

Factors that indicate with a binary value if this hashtag was seen or used dominantly were used on purpose, because ratio's can be ambiguous. For example, ratio of 40% seems very high, but there could be another hashtag that was seen 60% of the exposures. The same holds for small ratio's; a ratio of 10% seems small, but perhaps the user used this hashtag most frequently, because he also used 11 other hashtags too. So although a binary decision is strict (the second most dominant hashtag does not matter), the value 1 will always mean that this hashtag was seen or used most often (or is on a shared first place).

Lastly I added a factor (`followerCountMostPopularInfluencer`) that described how popular the most popular influencer (based on follower count)is that used the hashtag until this time window. It could be that users are more likely to adopt a hashtag that is used by a very influential user. The reason this factor was added and not the follower count of the influencer of the first exposure of the time window is two-fold. First, this factor will 'preserve' the value of the most popular influencer better than the follower count of the current influencer and I assume that a high follower count of any influencer that had exposed the user counts more than the exact follower count of the current influencer. Secondly, this preservation of the most popular influencer is helpful, because the data was filtered and only observations where the user used any relevant hashtag within the time window remained. This filter could have thrown out cases where the current influencer's follower count was very high and the fact that a very influential user had used this hashtag will not be taken into account anymore. The filter will be explained later more thoroughly.

Model set-up

The exact model used for analysis has two random effects: the user ID and the hashtag (the text of the hashtag). User ID is a random effect, because observations from the same user are probably correlated. Hashtag is a random effect, because it makes sense that a very frequently used hashtag behaves differently from a very infrequently used hashtag. The fixed predictors are these six factors:

- `userInfluencerCount`
- `uniqueInfluencers`

- `influencerLanguageIsDominantLanguageUser`
- `isDominantHashtag`
- `hashtagDominantPastUse`
- `followerCountMostPopularInfluencer`

The fixed effects need to be added as a slope to a random effect, when applicable. A factor could have a different effect for the different user ID's and /or for the different hashtags if the value of that factor is not mostly the same. This means that all factors were added as a slope for both random effects except: `userInfluencerCount` was not a slope for `userid` as a user will always have the same number of influencers and since I assume that frequently used or seen hashtags are frequent across observations `isDominantHashtag` and `hashtagDominantPastUse` were no slopes of `hashtag`.

Interactions were also added to the model: `userInfluencerCount` * `isDominantHashtag` and `userInfluencerCount` * `hashtagDominantPastUse`. These interactions were added to test if the effect network size might have for the different levels of `isDominantHashtag` and `hashtagDominantPastUse`. It could be there is no effect of network size when the hashtag is seen very little, e.g. users with large and small networks are not likely to adopt any hashtag that they did not see most often. It could also be that users with small networks are more likely to adopt a hashtag that they did not use dominantly before and users with large networks only use hashtags that they already used before.

Filters and normalization

The data contains $287,799$ observations and contained many cases where the user did not tweet anything between two exposures, or the user did tweet, but regarding something not related to the attacks in Brussels. As I want to test if users with smaller networks are more malleable observations where users did not use any relevant hashtag within the time window were discarded. It is difficult to see if a user decides to use this hashtag that he has been exposed with, if he did not use any hashtag whatsoever. Perhaps the user did not adopt the hashtag for other another reason, like not wanting to use a hashtag at all. This filter reduced the amount of observations to $44,027$ rows and from 979 unique users to 504 users. The reduction in users is caused by the fact that some of the original $76,211$ users that used one of the relevant 177 hashtags in the Streaming Data could be in that dataset, because they retweeted a relevant hashtag. These retweets are later not regarded as a use of a relevant hashtag by a user. It also occured that some users did use a relevant hashtag in some observations, but they were never exposed to that hashtag by any of their influencers.

All continuous factors (which are `userInfluencerCount`, `uniqueInfluencers` and `followerCountMostPopularInfluencer`) were log-transformed (base 2), because their distribution is skewed and I did not want the large outliers to have a disproportional effect. These three factors were also centered around 0 by subtracting the mean.

The predictor `hashtagDominantPastUse` is defined as a factor in the model. Another analysis with `hashtagDominantPastUse` as an ordened predictor showed that the relationship between this factor and 'adopted' is not linear. Therefore it was set to be a factor. `hashtagDominantPastUse` was releveled so 0 would be used as intercept. This way the results of this factor could be interpreted easier.

False convergence first mixed model

The default model as described above did not converge. This can be caused by many different reasons. For example, it is possible that there is no effect in the data whatsoever, the model cannot compute the effects of all slopes combined with every value of the random effects within the maximum number of iterations or it could be that one variable or slope causes the model to not converge. The influencer popularity factor seemed to have little influence on the likelihood of adoption based on the results of the model and seemed not to interact or interfere with influence of other factors. So a second analysis without `followerCountMostPopularInfluencer` was ran and the results of this analysis are listed in the Results section.

2.7.3 *Linear regression analysis*

The collapsed data file was used to do two linear regression models. A mixed effects model is not applicable for this dataset as it just contains one observation per user. The aim of these analyses is to test if social network size influences the number of hashtags the user uses or adopts. There is a difference between the number of hashtags the user adopts and the number he uses. Adopted hashtags is the number of unique hashtags the user adopted after been exposed to it. Used hashtags describes the number of unique hashtags the user used. A user could have invented a hashtag by himself or saw it elsewhere (for example, a Twitter user he does not follow used that hashtag, or he saw the hashtag on another social media platform). The value of this latter factor is therefore always higher than the number of adopted hashtags.

Two regression models were created; one with number of adopted hashtags as a dependent variable and one with the number of used hashtags. To prevent assigning effect size to the wrong predictor two other factors were added that could be influenced by influencer count and that could influence the number of adopted/used hashtags. Both models included these three predictors:

- The number of influencers the user has.
  The hypothesis is that people with smaller influencer counts will be use/adopt more different hashtags.
- The number of unique hashtags the user saw.
  It makes sense that the more unique hashtags a user is exposed to, the more he will use/adopt hashtags.
- The number of hashtags the user saw.
  My hypothesis is that the more hashtags a user sees, the more likely he is to use/adopt hashtags too.

RESULTS

3

This chapter lists and interprets the results from the two analyses as described in the Methods chapter.

3.1 RESULTS MIXED MODEL

The results of the linear mixed model show:

- A negative, significant effect of network size on adoption.
  ($\beta = -1.134e-03$, $SE = 1.827e-04$, $z = -6.203$, $p < 0.001$)
  The number of people a user follows influences the likelihood of adoption. This effect is negative, which means that the larger the network, the smaller the chance of adoption.

- A positive, significant effect of unique influencers on adoption.
  ($\beta = 4.118e-01$, $SE = 4.857e-02$, $z = 8.480$, $p < 0.001$)
  This positive effect means that likelihood of adoption increases as the number of unique influencers using the hashtag also increases.

- A positive, significant effect of dominant hashtag exposure on likelihood of adoption.
  ($\beta = 6.325e-01$, $SE = 9.526e-02$, $z = 6.640$, $p < 0.001$)
  When the hashtag was indeed the hashtag the user saw most often compared to other hashtags he has seen, likelihood of adoption increased.

- A positive, significant effect of influence language on adoption.
  ($\beta = 2.490e-01$, $SE = 6.776e-02$, $z = 3.675$, $p < 0.001$)
  The language of the tweet that contained the hashtag influences the likelihood of adoption. If this language is the same as the dominant language of the user, chances of adoption increase.

- A negative, significant effect of past dominant hashtag use (value -1) on adoption.
  ($\beta = -3.052e-01$, $SE = 7.997e-02$, $z = -3.816$, $p < 0.001$)
  This means that if the user already used another hashtag most often compared to when the user had not used a hashtag at all, likelihood of adoption decreases.

- A positive, significant effect of past dominant hashtag use value 1 on adoption.
  ($\beta = 1.687$, $SE = 1.183e-01$, $z = 14.262$, $p < 0.001$)
  When the user was already using this hashtag, chances of adoption increased.

- No interaction effects.
  The fact that we did not find any interactions mean that the factors by themselves only have an effect and that there is no additional effect when two factors are combined.

3.2 LINEAR REGRESSION MODEL

The collapsed data was used for two linear regression models. The results are very similar, so in order to make it easier to compare the β coefficients and p values the statistics are listed right underneath each other. The results of the linear regression models with the collapsed data show:

- A negative, significant effect of network size on the number of different hashtags used/adopted.
  (On used: $\beta = -1.048e-03$, $SE = 2.626e-04$, $t = -3.990$, $p < 0.001$)
  (On adopted: $\beta = -1.006e-03$, $SE = 2.332e-04$, $t = -4.314$, $p < 0.001$)
  This means that users with smaller networks are more likely to use/adopt more, different hashtags.

- A positive, significant effect of the number of different hashtags the user was exposed to on the number of different hashtags used/adopted.
  (On used: $\beta = 2.165e-02$, $SE = 7.872e-03$, $t = 2.751$, $p < 0.01$)
  (On adopted: $\beta = 3.367e-02$, $SE = 6.992e-03$, $t = 4.816$, $p < 0.01$)
  Users who have seen more, different hashtags tend to use/adopt more, different hashtags too.

- A positive, significant effect of the number hashtags a user has been exposed to on the number of different hashtags used/adopted.
  (On used: $\beta = 1.182e-03$, $SE = 2.731e-04$, $t = 4.329$, $p < 0.001$)
  (On adopted: $\beta = 1.018e-03$, $SE = 2.426e-04$, $t = 4.195$, $p < 0.001$)
  Users who have been exposed to more hashtags are more likely to use/adopt more, different hashtags.

The reason the two models are very similar in output is that the number of hashtags used and number of hashtags adopted is strongly correlated ($\rho = 0.951374$). The high correlation indicates that the number of used hashtags is almost the same as the number of adopted hashtags. This implies that users are not likely to use hashtags they have not been exposed to, i.e. they are not very likely to invent hashtags themselves or to propagate hashtags from other input sources on Twitter.

4 DISCUSSION

The aim of this Master thesis research project was to analyse the effect network size has on language malleability. In this study these terms are defined as the number of people a Twitter user follows and how likely that Twitter user is to copy hashtags from the people he is following, respectively. This study tests the hypothesis that people with a smaller network assign more weight to non-normative input and might therefore be more likely to propagate linguistic novelties. Novelties need to be propagated, because if just one person starts using a new word we cannot speak of language change.

By studying hashtag malleability on Twitter it was possible to control for different input channels and to work with real world data. Harvested data consists of tweets about the tragic terrorist attack in Brussels from users and the users they follow (also referred to as influencers).

4.1 TIME WINDOW DATA ANALYSIS

A mixed model linear regression analysis, with data that describes adoption behaviour of users between two consecutive exposures to the same hashtag, shows that the hypothesis is true; i.e. that users with more influencers are less likely to adopt hashtags from their influencers. The effect is very significant, but unfortunately it is difficult to say something about the size of the effect social network size has on adoption. Not because I used a log-transform for the factor network size, but because a mixed model assumed there are random effects and these random effects have different coefficients for every possible value of that random effect. Therefore it is not possible to construct the linear regression formula of a mixed model analysis and compute what an additional thousand influencers would do the the probability of adoption.

Other factors like the number of influencers who used the hashtag or the user's hashtag use in the past also significantly influenced the likelihood of adoption, but do not modulate the effect of number of influencers on adoption.

4.2 COLLAPSED DATA ANALYSIS

The second models show that in general, users with fewer influencers are more likely to use a higher proportion of the hashtags they have been exposed to than users with more influencers.

The most interesting result of the two models is that apparently the number of adopted hashtags is strongly correlated to the number of used hashtags. This finding implies that the initial reason to use Twitter data in order to capture for (what was then hoped) most of the exposure the Twitter users had, was correct.

4.3 GENERALIZABILITY TO LANGUAGE CHANGE

The findings of this study show that indeed people with smaller networks, i.e. less input sources, tend to be more malleable and propagate more, different output on Twitter. These findings may therefore provide insights in how it is possible that rare events of linguistic novelties can overcome the threshold problem and cause language to change. But are these findings generalizable to the way linguistic novelties that are not hashtags are propagated?

The first concern that I want to raise is the fact that only data from one topic (about something very extreme; a terrorist attack) was harvested and analysed. I chose this topic, because it was one of the only topics that would fit the criteria I drafted. One of these criteria was that data has to be collected right from the beginning, otherwise the different competing hashtags will already have converged. So perhaps, what I measured by making the decision to harvest data so close after the attacks happened, were novel hashtags being created and propagated for only a very short while. After a few weeks the fluctuation in popular hashtag will disappear and people will have conformed to only one or a few hashtags to describe this topic.

The convergence is even so extreme that after a while, the hashtags are barely used (unless there is something to remind people of the tragic day). So has the Twitter hashtag dictionary actually changed? I think that future research that wants to build on the results of this thesis should not focus on one likewise topic, but rather focus on general data. I would advise to harvest a large sample of Streaming data (serveral months at least). Then start looking for a few new hashtags that are not used in a very bursty manner (very high frequently used and after a few days not used anymore), but rather start slow, but will be used more frequently after a few weeks. Then it is possible to see if users who started propagating the hashtag in it's early stages might have small networks. Perhaps the hashtag will make a sudden jump in popularity, maybe retweets show who caused the sudden increase in popularity.

However, although I do think this topic might not have been the best decision, I do believe that the hypothesis is true (that people with less input sources assign more weight to the utterance of one person). This hypothesis resembles Nettle's proposed explanation of parochialism (Nettle, 1999). It sounds reasonable that people with small networks feel more parochialistic and like it that they share a special variant of language with each other.

4.4 CONCLUSION

To conclude, although there is still a lot that can be done in this field, this study provides insights as to why some people tend to be more malleable and propagate language changes more than others. This will help explain how rare new words or other linguistic novelties can cause language, used by a larger population, to change.

OPTIONS FOR DATA COLLECTION

A

Twitter offers three methods for data harvesting: the REST API, the Streaming API and the Firehose. Each of these three methods has its advantages and limitations. The Firehose is suited for companies or institutes with budgets to buy large quantities of data. This method was therefore not applicable. The other two options are free and were used for data acquisition. This section describes how the REST API and Streaming API work, starting with authentication and the returned data format. This information and more can be found in the API documentation of Twitter: https://dev.twitter.com

A.1 AUTHENTICATION

Both the REST and Streaming API (Application Programming Interface) require authentication in order to harvest data. Twitter uses an authentication model called OAuth and this model has two options: a user-application authentication and an application-only authentication. In the first case the data requester need to identify the application the request is coming from and the user he wishes to obtain data from. The application-only authentication can be used without user context, but not all methods work without user authentication. This project only uses the app-only identification and this required creating an application development account. This account (that is tied to my personal Twitter account) grants the possibility to enter three application names and each application name will yield a consumer key and a consumer secret key. The combination of app name and the two keys can be used to create an identification-token and this token can be sent along a request to Twitter.

A.2 RETURNED DATA STRUCTURE

Twitter returns output in `JSON` (JavaScript Object Notation) format when responding to a request in both the REST and Streaming API. `JSON` data is readable for humans as it has an intuitive notation for different data structures. Data is stored in <name : value> pairs. This explicit coupling of names to values makes interpretation by both humans and machines less error prone. Curly brackets surround objects with attributes and lists are indicated by square brackets. See Figure **??** for a small example of what the `JSON` notation for a tweet could look like (in reality a tweet may have up to 34 attributes). In this example an object is returned that contains a list of statuses. This list contains two tweets. A tweet object contains a timestamp, a text and a user

object. The user object contains a timestamp of when the account was created and a name.

```
{
   âĂIJstatusesâĂİ : [
   {
      âĂIJcreated_atâĂİ : âĂIJWed Jun 06 20:07:10 +0000 2012âĂİ,
      âĂIJtextâĂİ : âĂIJThis is an example tweet textâĂİ,
      âĂIJuserâĂİ : {
         âĂIJcreated_atâĂİ: âĂIJWed May 23 06:01:13 +0000 2007âĂİ,
         âĂIJnameâĂİ: âĂIJIris MonsterâĂİ
      },
   {
      âĂIJcreated_atâĂİ : âĂIJWed Jun 06 21:17:10 +0000 2012âĂİ,
      âĂIJtextâĂİ : âĂIJThis is a second tweetâĂİ,
      âĂIJuserâĂİ : {
         âĂIJcreated_atâĂİ: âĂIJWed May 23 06:01:13 +0000 2007âĂİ,
         âĂIJnameâĂİ: âĂIJIris MonsterâĂİ
      }
   ]
}
```

Figure A.1: A small example of a TweetArray

A.3 REST API

The REST API (REpresentational State Transfer Application Programming Interface) can be used to make REST requests. As is common in REST APIs it is possible to perform get and post requests. A get-request will retrieve data and a post-request will send data to Twitter, for example posting a tweet on behalf of an authenticated user. I never used any post-request as they did not apply to this project, so I will not elaborate more on this type of request.

The REST API is limited by rate limits, a maximum number of returned objects per request and a maximum number of requests. The rate limits mean that there is a certain number of requests the app credentials can make per query. An example: The goal is to receive the post history of 100 users. The `GET statuses/user_timeline` request has a rate limit of 300 requests per 15-minute window. The maximum number of posts that can be received per request is 200 and per user it is possible to get his

latest 3200 posts. This means that, if all 100 users have more than 3200 posts and the goal is to get as much as possible, it takes 16 requests to retrieve the maximum amount of posts of one user. It takes 1600 requests to get the history of all users. Let us assume that it takes the program 15 minutes to make 300 requests (the computer can maximally handle what the rate limit also prescribes). Due to the rate limits this will take one hour and 20 minutes.

My experience is that doing the requests and processing the data do not take long, so my program would make requests for 1 minute and then wait 14 minutes due to the rate limit. I have made this waiting period shorter by creating six app accounts (with a second personal Twitter account), so I would be able to harvest six times as much within the same period of time. For this particular type of request this means that I can now harvest the tweet history of 100 users within 15 minutes and still not exceed the rate limit of the sixth app account. This decision was not completely risk free. Nowhere in the documentation of Twitter is mentioned that it is not allowed to use multiple authentication keys for one app. However, abusing Twitter's API may lead to suspension or terminating access. Perhaps it was possible to add another 3 keys, which would speed up harvesting even more, but the risk of banning was too great to try that out.

A.4 HARVESTING LISTS AND TIMELINES WITH THE REST API

JSON objects that Twitter returns for a request often contain cursor information or search metadata (see Figure 2.1). Cursor information can be used when navigating through a list that is longer than what is returned in one request. For example: When calling GET friends/ids method a `UserIDArray` is returned. This object contains an array of user ID's and cursor information. This particular method will return a maximum count of 5000 user ID's. When a user follows more people, the cursor can be used for the subsequent `GET friends/ids` method to get the next piece of the list.

Twitter has made a special exception for lists that contain timelines, like the GET statuses/lookup request. To avoid receiving duplicate tweets or missing tweets it is possible to add a `max id` or `since id` to methods where timelines are requested.

A.5 STREAMING API

The Streaming API is also restricted like the REST API, but in a different manner. When an application (that has been authenticated) calls the Streaming API returns a random sample of all tweets posted in real time from around the world. Twitter does not document a precise percentage and online sources describe percentage estimations that vary greatly. On average I would receive around 80 tweets per second, but these include partial tweets or delete notifications.

I used both APIs for harvesting data and created my own Java code to communicate with the APIs. All code for this research project can be found on GitLab, see Appendix G.

DESCRIPTION FILE SYNTAX

B

Here is the ANTLR4 grammar for a description file:

```
grammar Description;

@header
{
package nl.ru.ai.twiapigen.description;
}

// starting point for parsing a Description file
compilationUnit
:    typeDeclaration EOF
;

typeDeclaration
:    Comment? Identifier body
;

body
:    '{' fieldDeclaration* '}'
;

fieldDeclaration
:    Comment? sql? key? type row? fieldName ';'
;

fieldName
:    Identifier
;

type
:    Identifier
;

row
: '[]'
;
```

```
sql
: 'SQL'
;

key
: 'KEY'
;

Identifier
:   JavaLetter JavaLetterOrDigit*
;

fragment
JavaLetter
:   [a-zA-Z$_]
;

fragment
JavaLetterOrDigit
:   [a-zA-Z0-9$_]
;

//
// Whitespace and comments
//

WS  :  [ \t\r\n\u000C]+ -> skip
;

Comment
:   '/*' .*? '*/'
;

LINE_COMMENT
:   '//' ~[\r\n]* -> skip
;
```

C MYSQL DATABASE

The first version of TwiApiGen could process Twitter data and allowed me to call methods to access the information I needed. It was necessary to store data in a structure that is easy to access, fast and can be queried in a complex and structured fashion. Therefore, I chose to create a MySQL (My Structured Query Language) database.

MySQL is a database management system that consists of one or more tables. Every table consists of a number of columns and a column usually has a name and a type for the values in the column. Every row describes a data entry and fills one value for every column. A column can be left empty. A column cannot be left empty if it is an identifier for the rows. It is common to use an incrementing integer as the identifier for a row in a table, but for instance a name can also be used as long as an identifier when it is unique for every row. One of the features of MySQL is that these identifiers can be used to connect a row to data from another table. The connections are called foreign keys and are also used in the database structure of the Twitter data.

I added the possibility to add the word SQL before the type of a field in the description files, which indicates that this field needs to be added to a MySQL database structure.

C.1 java TYPES TO sql

Since the classes were generated based on the description files, I also wanted to generate the database access routines and a database template based on these files. To do so, I extended the ANTLR grammar so that the keyword SQL could be added to fields in the description files. Placing SQL in front of a field type specifies that this field should be part of the database. In some cases it was necessary to indicate which field should be used for identification of that object in the database. The tag KEY denotes the identifying fields. For Classes without a KEY field, an implicit auto incrementing integer identifier is created.

Besides the distinct Java Class that TwiApiGen creates it also creates a MySQL table template, e.g. a table 'Tweet' and table 'User'. A table contains rows per data entry and a column for every field. As with the Java objects, the original JSON basic data types can be easily mapped to SQL native data types like numbers or strings. The value in the corresponding column would be the value of the field. If the field is a nested object, the value is an identification key pointing to the place where that object is stored. See figure C.1 for an example of a generated table.

```
#
# SQL table definition for Contributor generated by
# TwiApiGen version 0.1 on Tue Mar 15 12:41:51 CET 2016
#
CREATE TABLE Contributor
(
  #
  # Automatically created auto-increment key,
  # not part of Twitter API
  #
  `key` INT NOT NULL AUTO_INCREMENT,
  #
  # The integer representation of the ID of the user who
  # contributed to this Tweet.
  #
  `id` BIGINT,
  #
  # The string representation of the ID of the user who
  # contributed to this Tweet.
  #
  `idStr` TEXT CHARACTER SET utf8mb4,
  #
  # The screen name of the user who contributed
  # to this Tweet.
  #
  `screenName` TEXT CHARACTER SET utf8mb4,
  PRIMARY KEY (`key`)
) ENGINE=INNODB;
```

Figure C.1: Example generated table by `TwiApiGen`

TwiApiGen generates special tables for arrays, one for each array the data contains. For example, both the class Hashtag and the class Symbol contain a LongArray. In this particular case it happens to be that both are for indices (indicating where the hashtag or symbol is within the tweet text). However I did not create a general indices table that would store them both. The database contained a table for SymbolIndices and one for HashtagIndices. Tables for arrays consist of two columns: parent and child. So the first column contains an id to the object that the array belongs to (for example a tweet id) and the second column contains either an element from the list or the id of an element from the list. So for every element from a list a row would be added to the table.

In practice it happened that certain objects did not have an id from Twitter or an id from Twitter was not called ID (like userid). To solve these issues I added the keyword KEY to the description file grammar. If a field has that keyword, TwiApiGen knows that that field should be used in the SQL database as the unique identifier. It would happen that no field had that keyword and that object had to be linked to another object, like in the case of the bounding box (this is part of the object Places and Places belongs to a Tweet). TwiApiGen always first inserts nested structures before inserting an object, so first the Bounding Box would be inserted. The index in the table of that object would then serve as the identifier that the parent contains.

See figure C.2 for the full EER diagram of the database. Here you can see links between the tables. These are called foreign keys. It a good habit to use foreign keys wherever possible, because MySQL is then able to report inconsistencies. For example, it will raise a red flag when you want to delete something that is linked to from somewhere else. And, when deleting the parent object, it will delete everything that links to that object. However, with this dataset I switched off foreign keys checking, because there are circular references in the database. For example a Tweet contains a User object, but User also contains a Tweet object (the latest post). Although there are methods to deal with circularity, implementing these methods would increase the complexity of TwiApiGen, so foreign key checking is disabled.

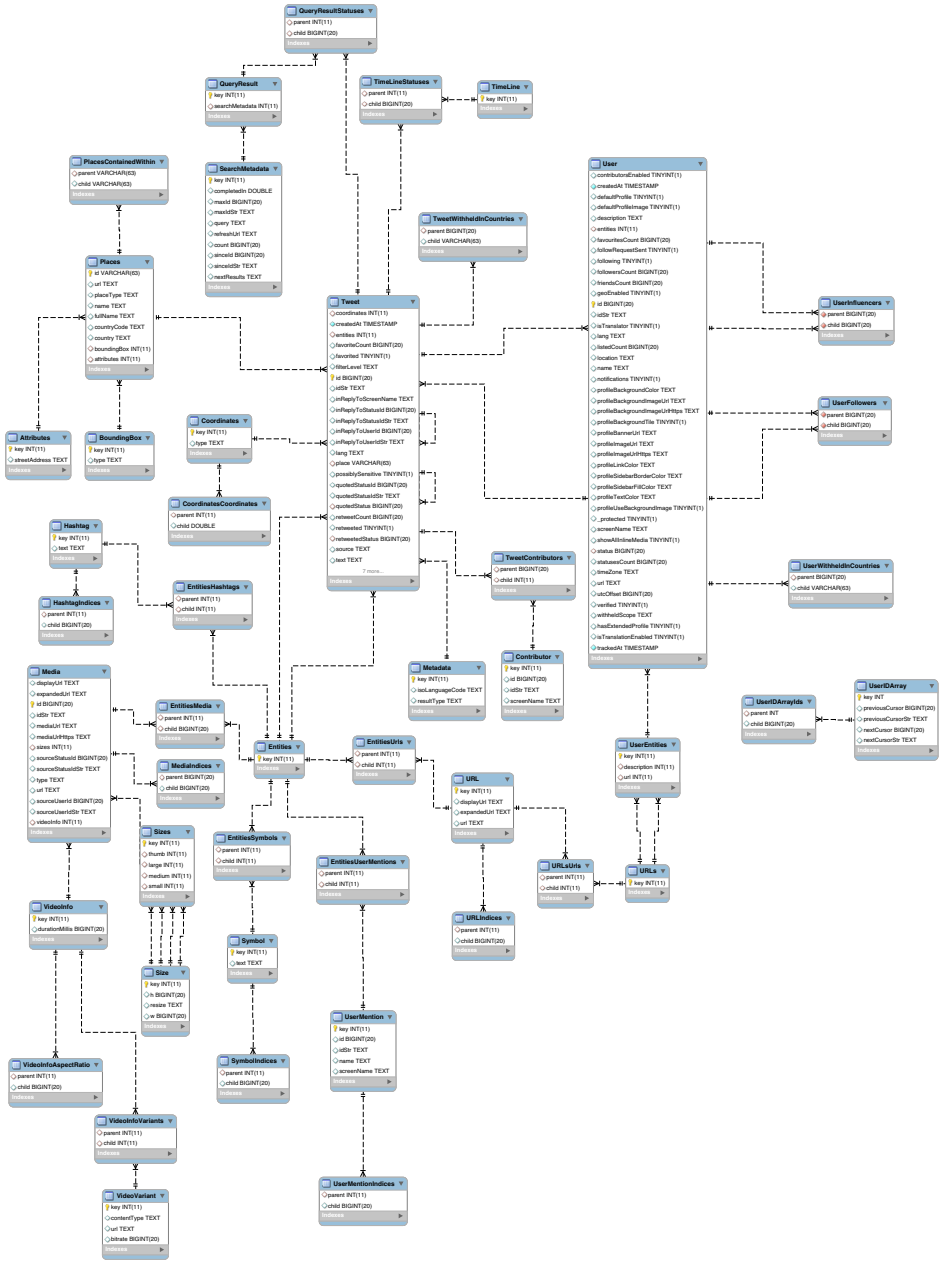


Figure C.2: EER diagram of database

SAMPLE GENERATED TWITTER CLASS

D

```
package nl.ru.ai.twitter.api;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;
import java.util.Arrays;
import java.util.Date;
import java.util.Locale;
import java.text.SimpleDateFormat;
import java.text.ParseException;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.JSONValue;
/*
** Class Hashtag generated by TwiApiGen version 0.1
** on Sun Jan 10 22:49:35 CET 2016
** Represents hashtags which have been parsed out of the Tweet text.
*/
public class Hashtag
{
  private static SimpleDateFormat dateFormat = new
      SimpleDateFormat("EEE MMM dd HH:mm:ss Z yyyy", Locale.US);
  private static Set<String> knownFields=new
      HashSet<String>(Arrays.asList(new String [] {"indices","text"}));
  private LongArray indices;
  private String text;
  private Integer key;
  /**
   * Constructor
   */
  public Hashtag(LongArray indices,String text)
  {
```

```
    this.indices=indices;
    this.text=text;
  }
  /**
   * Parse function
   * @param object object to parse
   * @return Hashtag object
   * @throws ParseException
   */
  public static Hashtag parse(Object object) throws ParseException
  {
    if(object==null)
      return null;
    JSONObject o=(JSONObject)object;
    LongArray indices=LongArray.parse(o.get("indices"));
    String text=(String)(o.get("text"));
    for(String key:(Set<String>)o.keySet())
      if(!knownFields.contains(key))
      {
        System.err.println(object);
        System.err.printf("Found field '%s' for type '%s' which is not
            in description file\n",key,"Hashtag");
        knownFields.add(key);
      }
    return new Hashtag(indices,text);
  }
  /**
   * Parse function
   * @param string string to parse
   * @return Hashtag object
   * @throws ParseException
   */
  public static Hashtag parse(String string) throws ParseException
  {
    return parse(JSONValue.parse(string));
  }
  @Override
  public String toString()
  {
    return toJSON().toString();
```

```
}
/**
 * Reconstruct JSON object from this object.
 * @return object in JSON format
 */
public JSONObject toJSON()
{
  JSONObject object=new JSONObject();
  if(indices!=null)
    object.put("indices",indices);
  if(text!=null)
    object.put("text",text);
  return object;
}
/**
 * INSERT/UPDATE rows into SQL table
 * @param connection connection to SQL database
 * @param parent id of parent
 * @throws SQLException
 */
public void insertUpdateRowIndices(Connection connection, Integer
    parent) throws SQLException
{
  for(Long child : indices)
  {
    PreparedStatement
        statement=connection.prepareStatement(String.format("INSERT
        INTO HashtagIndices (parent,child) VALUES (?,?)"));
    int arg=1;
    statement.setInt(arg++,parent);
    statement.setLong(arg++,child);
    statement.executeUpdate();
  }
}
/**
 * INSERT/UPDATE into SQL table
 * @param connection connection to SQL database
 * @throws SQLException
 */
public void insertUpdate(Connection connection) throws SQLException
```

```
{
  if(key==null)
  {
    StringBuffer select=new StringBuffer();
    if(text!=null)
      select.append("text=? AND ");
    else
      select.append("text IS NULL AND ");
    if(select.length()!=0)
    {
      select.setLength(select.length()-5);
      PreparedStatement
          statement=connection.prepareStatement(String.format("SELECT
          `key` FROM Hashtag WHERE %s",select));
      int arg=1;
      if(text!=null)
        statement.setString(arg++,text);
      ResultSet resultSet=statement.executeQuery();
      if(resultSet.next())
        key=resultSet.getInt("key");
      resultSet.close();
      statement.close();
    }
  }
  if(key==null)
  {
    PreparedStatement
        insertStatement=connection.prepareStatement(String.format("INSERT
        INTO Hashtag VALUES ()"));
    insertStatement.executeUpdate();
    insertStatement.close();
    PreparedStatement
        selectStatement=connection.prepareStatement("SELECT
        LAST_INSERT_ID() AS `key` FROM Hashtag");
    ResultSet selectSet=selectStatement.executeQuery();
    if(selectSet.next())
      key=selectSet.getInt("key");
    selectSet.close();
    selectStatement.close();
  }
```

```
    if(indices!=null)
      insertUpdateRowIndices(connection,getKey());
    StringBuffer update=new StringBuffer();
    if(text!=null)
      update.append("text=?,");
    if(update.length()>0)
    {
      update.setLength(update.length()-1);
      PreparedStatement
          statement=connection.prepareStatement(String.format("UPDATE
          Hashtag SET %s WHERE `key`=?",update));
      int arg=1;
      if(text!=null)
        statement.setString(arg++,text);
      statement.setInt(arg++,key);
      statement.executeUpdate();
      statement.close();
    }
  }
  /**
   * An array of integers indicating the offsets within the Tweet text
   * where the hashtag begins and ends.
   * The first integer represents the location of the # character in
   * the Tweet text string.
   * The second integer represents the location of the first character
   * after the hashtag.
   * Therefore the difference between the two numbers will be the
   * length of the hashtag name plus one (for the âĂŸ#âĂŹ character).
   * @return the indices
   */
  public LongArray getIndices()
  {
    return indices;
  }
  /**
   * Name of the hashtag, minus the leading âĂŸ#âĂŹ character.
   * @return the text
   */
  public String getText()
  {
```

```
    return text;
  }
  /**
   * This field is added for SQL and was not part of the description
   * file
   * @return the key
   */
  public Integer getKey()
  {
    return key;
  }
}
```

E

#airport
#attack
#attacks
#attentats
#banislam
#belgi
#belgian
#belgica
#belgique
#belgium
#belgiumairport
#belgiumattack
#belgiumattacks
#belgiumstrong
#belgiumunderattack
#belguim
#blast
#bomb
#bombing
#bombings
#borders
#breakingnews
#bru
#brublasts
#bruessel
#bruessels
#brus
#bruselas
#bruss
#brusse
#brussel
#brusselairport
#brusselattack
#brusselattacks
#brussells
#brussels
#brusselsa
#brusselsairport
#brusselsat
#brusselsatta
#brusselsattack
#brusselsattacks
#brusselsblast
#brusselsblasts
#brusselsbomb
#brusselsbombing
#brusselsexplosions
#brusselslift
#brusselslockdown
#brusselsmetro
#brusselssubway
#brussles
#brusslesattacks
#brusssels
#brux
#bruxelles
#bruxellesattack
#bruxellesmabelle
#bruxells
#deportallmuslims
#dontstopislam
#espacepourlavie
#explosion
#explosions
#fuckislam
#fuckterrorists
#heartgoesouttobrussels
#icantbelieveijustsaw
#ikwilhelpen
#ilovebelgium
#ilovebruxelles
#isis
#islam
#islamholdspeace
#islamispeace
#islamistheproblem
#islamkills
#islamophobia
#istandwithbrussels
#jesuisbelge
#jesuisbrussel
#jesuisbrussels
#jesuisbruxel
#jesuisbruxelle
#jesuisbruxelles
#jmahers
#lockdown
#lovebrussels
#loveislam
#lovemuslims
#maalbeck
#maalbeek
#maalbek
#maelbeek
#maketheworldsafeagain
#malbeek
#mannekenpis
#metro
#metrostation
#multipleattacks
#muslimneighborhood
#muslims
#muslimsforpeace
#muslimslivematters
#noalterrorismo
#notallmuslims
#peace
#peaceforbrussels

#peaceonearth	#porteouverte	#pray
#prayers	#prayersforbelg	#prayersforbelgium
#prayersforbru	#prayersforbrussels	#prayf
#prayforbel	#prayforbelgium	#prayforbruessels
#prayforbruselas	#prayforbrusells	#prayforbrusels
#prayforbrussel	#prayforbrusseles	#prayforbrussels
#prayforbrussles	#prayforbruxelle	#prayforbruxelles
#prayforourworld	#prayforpeace	#prayfort
#prayforth	#prayforthewholeworld	#prayfortheworld
#prayforworldspeace	#prayingforbelgium	#prayingforbrussels
#solidaritywithbrussels	#somethingneedstochange	#standwithbrussels
#startislam	#station	#staysafebrussels
#staystrongbelgium	#staystrongbrussels	#stopblamingislam
#stopbombingnow	#stophate	#stopis
#stopisalm	#stopisis	#stopislam
#stopislamophobia	#stopislamphobia	#stopreligion
#stopterrorism	#subway	#suicideattack
#terreur	#terrorattack	#terrorinbrussels
#terrorism	#terrorismhasnoreligion	#terroristattacks
#terrorists	#thisistheworldwelivein	#thoughtsandprayers
#tousensemble	#waronmuslims	#whenwillitstop
#whitegenocide	#zavantem	#zavente
#zaventem	#zaventemairport	#zaventem

Table E.1: Unique hashtags

CSV TABLE

F

name	type	Description
hashtag	String	The text of the hashtag.
influenceTimestamp	Date	When the user was exposed to the hashtag. This is the start of the timewindow.
influenceTweetID	Long	The ID of the tweet that contains the hashtag. This tweet will be referred to as *this exposure*.
influenceIsReply	Binary	Equals 1 if the influence tweet is a reply.
influenceReplyToUser	Binary	Equals 1 if the influence tweet is a reply to the user.
influenceLanguage	String	Language of the influence tweet according to Twitter.
influenceIsQuote	Binary	Equals 1 if the influence tweet is a quote.
influenceQuoted TweetText	String	Contains, if influence tweet is a quote, the text of the tweet that is quoted.
influenceQuotedUser	Binary	Equals 1 if the influence tweet quotes the user.
influenceRetweeted Count	Integer	Number of times the influence tweet is retweeted by Twitter users.
influenceTweetText	String	Text of the influence tweet.
influenceIsTruncated	Binary	Equals 1 is the influence tweet is truncated.
influencer FavoriteCount	Integer	Number of times the influence tweet is favorited by Twitter users.
influencer FollowersCount	Long	Number of followers the influencer has.
influencer InfluencersCount	Long	Number of influencers the influencer has.
influencerUserID	Long	The ID of the influencer.
influencer ScreenNameUser	String	The screenname of the influencer.
influencerTweetCount	Long	Number of tweets the influencer posted on Twitter since he created his account.
influenceNumberOf HashtagsInTweet	Integer	Number of hashtags that were in the text of the influence tweet.
userId	Long	The ID of the user.
userScreenName	String	The screenname of the influencer.
userInfluencerCount	Long	Number of influencers the user has.
userFollowersCount	Long	Number of followers the influencer has.

usersFavoritesCount	Integer	Number of times the user has been favorited by Twitter users.
usersTweetCount	Long	Number of tweets the user posted on Twitter since he created his account.
ratioHashtag DominantHashtag	Double	Number of times the user has been exposed to this hashtag divided by the number of times the user has been exposed to any hashtag until and including this exposure.
isDominantHashtag	Binary	Equals 1 if this hashtag is the hashtag the user has been exposed to the most until and including this exposure (also in case of a shared first place with other hashtags).
nextInfluence Timestamp	Date	The next time the user is exposed to the same hashtag by the same or another influencer. This is the end of the timewindow.
tagExposure	Integer	Number of times the user is exposed to this hashtag until and including this exposure.
uniqueInfluencers	Integer	Number of influencers who used this hashtag until and including this exposure.
followerCountMost PopularInfluencer	Long	Number of followers of the influencer with the highest number of followers of all influencers who have used this hashtag until and including this exposure.
medianFollowerCount Influencers	Long	The median of the follower counts of influencers who have used this hashtag until and including this exposure.
windowSize	Long	Number of milliseconds between the timestamps of the start and end of the time window.
adopted	Binary	Equals 1 if the user has used this hashtag within the time window at least once.
usedWithinWindow	Integer	Number of times the user has used this hashtag within the time window.
userUsedHashtag Dominant WithinWindow	Binary	Equals 1 if the user used this hashtag the most (also in case of a shared first place) within the time window.
usedHashtag BeforeInfluence	Binary	Equals 1 if the user had used the hashtag before this exposure.
usedOtherHashtags BeforeInfluence	Binary	Equals 1 if the user had used other hashtags before this exposure.

countUsedOther RelevantHashtag WithinWindow	Integer	Number of times the user used any other relevant hashtag within the time window.
tweetedWithoutAny RelevantHashtag WithinWindow	Binary	Equals 1 if the user has tweeted without any relevant hashtag within the time window.
hashtagDominant PastUse	-1/0/1	Equals 1 if the user used this hashtag mostly (also in case of a shared first place) before this exposure. Equals 0 if the user had not used any relevant hashtag yet before this exposure. Equals -1 if the user mostly used another relevant hashtag before this exposure.
countUserUsed HashtagsBeforeWindow	Integer	Number of times the user used hashtags before this time window.
countUserUsedUnique HashtagsBeforeWindow	Integer	Number of unique hashtags the user used before this time window.
userUsedKeyWord WithinWindow	Binary	Equals 1 if the user used one of the hashtags without the #-symbol within this time window.
userUsedAnyRelevant HashtagWithinWindow	Binary	Equals 1 if the user used any relevant hashtag within this time window.
userOnlyTweetedAbout OtherStuffWithin Window	Binary	Equals 1 if the user did not use any relevant hashtag and did not use any relevant hashtag as a keyword (so without the #-symbol.)
countHashtags CompleteHistory	Integer	Number of hashtags the user used within three weeks after the attacks happened.
countUniqueHashtags CompleteHistory	Integer	Number of unique hashtags the user used within three weeks after the attacks happened.
influenceLanguageIs DominantLanguageUser	Binary	Equals 1 if the language of this exposure is also the dominant language of the user (also in case of a shared first place).
influenceLanguageIs DominantLanguageUser WithinWindow	Binary	Equals 1 if the language of this exposure is the same as the dominant language of the user's tweets he posted within this time window (also in case of a shared first place).
userUsedLanguage OfInfluenceWithin Window	Binary	Equals 1 if the user has used the language of this exposure at least once within this time window.
allLanguagesGeneral	String	All languages and frequency counts the user used within 3 weeks after the attacks.

allLanguagesBeforeNextInfluence	String	All languages and frequency counts the user used within this time window and before this exposure.
amountUserMentionedInfluencer	Integer	Number of times the user mentioned the influencer of this exposure.
firstAdoptionTimestamp	Date	First adoption fields are filled if the user used this hashtag within the window and will describe the (first, if there are multiple tweets the user posted within this window with this hashtag) tweet the used posted with this hashtag. This is the time stamp.
firstAdoptionLanguage	String	Language of the first adoption.
firstAdoptionTweetID	Long	The ID of first adoption tweet.
firstAdoptionTextTweet	String	Text of the first adoption tweet.
firstAdoptionNumberOfHashtagsInTweet	Integer	Number of hashtags the first adoption tweet contains.
firstAdoptionIsTruncated	Binary	Equals 1 if the first adoption tweet is truncated.
firstAdoptionRetweetedCount	Integer	Number of times the first adoption is retweeted by Twitter users.
firstAdoptionIsReply	Binary	Equals 1 if the first adoption tweet is a reply.
firstAdoptionIsReplyToInfluence	Binary	Equals 1 if the first adoption tweet is a reply to the influencer of this exposure.
firstAdoptionIsQuote	Binary	Equals 1 if the first adoption is a quote.
firstAdoptionQuotesInfluencer	Binary	Equals 1 if the first adoption quotes a tweet from the influencer of this exposure.
firstAdoptionQuotedTweetText	String	The text of the first adoption tweet.

Table F.1: CSV table

CODE

Code used for this project can be found on my GitLab repository.
The TwiApiGen project can be found on:
https://gitlab.socsci.ru.nl/twitterthesisproject/twiapigen.git
The Stanford parser projects can be found on:
https://gitlab.socsci.ru.nl/twitterthesisproject/stanfordparser.git
Code for the relevance weighting can be found on:
https://gitlab.socsci.ru.nl/twitterthesisproject/twitterwordcloud.git
The code I used to harvest and process data can be found on:
https://gitlab.socsci.ru.nl/twitterthesisproject/twitterharvest.git

REFERENCES

Baregheh, A., Rowley, J., & Sambrook, S. (2009). Towards a multidisciplinary definition of innovation. *Management Decision*, *47*(8), 1323–1339. doi: 10.1108/00251740910984578

Bates, D., Mächler, M., Bolker, B., & Walker, S. (2014). Fitting linear mixed-effects models using lme4. *arXiv preprint arXiv:1406.5823*.

Croft, W. (2000). *Explaining Language Change. An Evolutionary Approach.* Pearson Education.

Efron, B., & Morris, C. N. (1977). *Stein's paradox in statistics.* WH Freeman.

Fang, Y. (2016). *JSON simple, a toolkit for Java.* Retrieved from https://code.google.com/archive/p/json-simple/ (Last accessed 29 February 2016)

Hargadon, A. (2003). *How Breakthroughs Happen: The Surprising Truth About How Companies Innovate.* Harvard Business Review Press.

Harrington, J., Palethorpe, S., & Watson, C. I. (2000). Does the Queen speak the Queen's English? *Nature*, *408*(6815), 927–928.

Jiang, J. (2007). *Linear and generalized linear mixed models and their applications.* Springer Science & Business Media.

Keller, R. (1995). *On Language Change: The Invisible Hand in Language* (1st ed.). Routledge.

Klein, D., & Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1* (pp. 423–430). Stroudsburg, PA, USA: Association for Computational Linguistics. doi: 10.3115/1075096.1075150

Laird, N. M., & Ware, J. H. (1982). Random-effects models for longitudinal data. *Biometrics*, 963–974.

Lev-Ari, S. (2016). *Talking to fewer people leads to having more malleable linguistic representations.*

Manning, C. D., Bauer, J., Finkel, J., Bethard, S. J., Surdeanu, M., & McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 55–60.

Nettle, D. (1999). Using Social Impact Theory to simulate language change. *Lingua*, *108*(2-3), 95–117.

Parr, T. (2013). *The definitive antlr 4 reference* (2nd ed.). Pragmatic Bookshelf.

Pontoriero, D., & Gillie, E. (2012). Binders Full Of Women: Convergence of Independently-Generated Hashtags on Twitter. In *Class on social and information network analysis* (pp. 1–8).

Robertson, S., & Jones, K. (1976). Relevance weighting of search terms. *J. Am. Soc. Inf. Sci.*, *27*(3), 129–146. doi: 10.1002/asi.4630270302

Romero, D. M., Meeder, B., & Kleinberg, J. (2011). Differences in the mechanics of information diffusion across topics: idioms, political hashtags, and complex contagion on twitter. In *Www'11 proceedings of the 20th international conference on world wide web* (pp. 695–704). Hyderabad, India: ACM.

Sapir, E. (1921). *Language: An introduction to the study of speech.* New York : Harcourt, Brace and Company.

Yang, L., Sun, T., Zhang, M., & Mei, Q. (2012). We know what@ you# tag: does the dual role affect hashtag adoption? In *Www'12 proceedings of the 21st international conference on world wide web* (pp. 261–270). Lyon, France: ACM.

Yu, C. T., & Salton, G. (1976, January). Precision Weighting; An Effective Automatic Indexing Method. *J. ACM*, *23*(1), 76–88. doi: 10.1145/321921.321930