UPPSALA
UNIVERSITET

TVE-F 18 001
Examensarbete 15 hp
Maj 2018

Creating Interactive Visualizations for Twitter Datasets using D3

Olof Björck

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Creating Interactive Visualizations for Twitter Datasets using D3

Olof Björck

Project Meme Evolution Programme (Project MEP) is a research program directed by Raazesh Sainudiin, Uppsala University, Sweden, that collects and analyzes datasets from Twitter. Twitter can be used to understand how ideas spread in social media. This project aims to produce interactive visualizations for datasets collected in Project MEP. Such interactive visualizations will facilitate exploratory data analysis in Project MEP. Several technologies had to be learned to produce the visualizations, most notably JavaScript, D3, and Scala. Three interactive visualizations were produced; one that allows for exploration of a Twitter user timeline and two that allows for exploration and understanding of a Twitter retweet network. The interactive visualizations are accessible as Scala functions and in a website developed in this project and uploaded to GitHub. The interactive visulizations contain some known bugs but they still allow for useful exploratory data analysis of Project MEP datasets and the project goal is therefore considered met.

Handledare: Raazesh Sainudiin
Ämnesgranskare: Ocean Cheung
Examinator: Martin Sjödin
ISSN: 1401-5757, TVE-F 18 001

Populärvetenskaplig sammanfattning

Twitter används ibland av forskare för att undersöka hur idéer sprids i sociala medier. Forskaren Raazesh Sainudiin på Uppsala Universitet driver ett forskningsprojekt han kallar Project Meme Evolution Programme (Projekt MEP) som går ut på att samla massor av tweets och information från Twitter och sedan undersöka den informationen. Till exempel kan det vara intressant att titta på hur riksdagsledamöter använder sig av Twitter innan ett val. Det, bland annat, har gjorts i Projekt MEP. När man samlar massor av information, flera miljoner tweets i Projekt MEP, kan det vara svårt att få en överblick av all den information man samlat. Då kan det vara bra att använda så kallade visualiseringar. Visualiseringar är bara ett samlingsnamn för små bilder som sammanställer information, till exempel vanliga diagram. Om man vill gå ännu längre kan man göra interaktiva visualiseringar som är visualiseringar man kan klicka på och utforska. Interaktiva visualiseringar är användbara när man vill presentera riktigt mycket information på ett enkelt sätt. Det här projektet har gått ut på att göra interaktiva visualiseringar där det går att utforska den information som samlas i Projekt MEP. I projektet gjordes tre interaktiva visualiseringar; en där man kan utforska en Twitteranvändares historik och två där man kan utforska hur olika Twitteranvändare har interagerat med varandra. På det stora hela fungerar de interaktiva visualiseringarna bra och är användbara för att utforska information som samlas i Projekt MEP men det finns också små skrupler som skulle kunna åtgärdas och några idéer till hur de interaktiva visualiseringarna kan förbättras. Projektet är överlag lyckat och nådde de mål som skulle uppnås.

Contents

1 Introduction

1.1 Background

Twitter [1] is a valuable research environment to understand how ideas spread in social media today. Project Meme Evolution Programme (Project MEP) [2] is a research program directed by Raazesh Sainudiin, Department of Mathematics, Uppsala University, Sweden, focused on providing researchers in the Humanities with user data from Twitter for their research. Research that Project MEP have been or is currently involved in is for example monitoring and exploration of Members of Parliaments' Twitter activity prior to general elections in New Zeeland, The United Kingdom, and Sweden.

Data from Twitter can be collected free of charge using Twitter's application programming interface (Twitter's API). The data that can be collected from Twitter can be very extensive; most importantly for this project is that specific Twitter posts and which Twitter users have interacted with each other can be collected. A lot of code necessary to collect and store big amounts of Twitter data have already been implemented within Project MEP. This project relies heavily on code implemented by Joakim Johansson in his Master's thesis [3] within Project MEP at Uppsala University.

Beyond collecting and storing big amounts of Twitter data, methods for analyzing collected Twitter data have been developed within Project MEP. Such analysis is for example computing how many intermediary Twitter users are needed to find an interaction path between two Twitter users. However, no non-trivial visualizations of collected Twitter data have been developed within Project MEP. Visualizations are usually a part of the data science process and are a common and useful way to gain a general understanding of data before conducting analysis.

1.2 Interactive visualization

An interactive visualization [4] is any type of visualization that allows for dynamic presentation of data. In contrast to regular static visualizations such as plots and diagrams found in research papers, an interactive visualization requires a computer and as it's name suggests, interaction and not just observation. In an interactive visualization, some sort of response is given by the visualization as a user interacts with it. For example, if a user hovers over a data entry, additional information not shown until then can be displayed, e.g. mean values, names, and counts. In this way, a lot more data can be incorporated in an interactive visualization than in a static visualization because in the interactive visualization, data can be hidden unless needed. Therefore, interactive visualizations are a way to quickly and conveniently explore data that would not be possible to display in a static visualization or would require additional explanations and tables, thus making understanding the data slow and tedious.

1.3 Objective

This project aims to develop interactive visualizations for the Twitter datasets that are collected in Project MEP. Specifically, this project aims to produce interactive visualizations that allow for

- exploration of a user timeline;
- exploration of a Twitter network; and
- understanding of a Twitter network.

Such interactive visualizations are expected to provide a quick and convenient way to gain understanding of Project MEP datasets.

2 Theory

2.1 Technologies and programming languages

2.1.1 Scala

Scala [5] is a programming language originally designed to improve on Java. Scala combines object oriented programming with some functional programming capabilities. Scala is suitable for big data applications and most code within Project MEP prior to this project is written in Scala.

2.1.2 HTML

Hypertext Markup Language (HTML) [6] is the language used to define a website. The HTML code in a website can be thought of as the website's skeleton, defining the website's core structure.

2.1.3 CSS

Cascading Style Sheets (CSS) [7] is the language used to define the styling of a website's HTML code. The main point of CSS is to provide a separation between the website's structure, i.e. the HTML code, and the website's styling, i.e. the CSS code.

2.1.4 JavaScript

JavaScript [8] is the programming language used to create interactive web pages. HTML and CSS alone would provide a rather static, "read-only" website but JavaScript allows for handling user events and real-time modification of the website content. Unlike HTML and CSS, JavaScript is a proper programming language and not just a language used for structural definitions. JavaScript is also used for non-web programs.

2.1.5 SVG

Scalable Vector Graphics (SVG) [9] is a common vector image format for websites. Interactive visualizations can be created using SVG.

2.1.6 CSV

Comma-separated values (CSV) [10] is a common data format where a comma is used to separate data values. The CSV data format can be thought of as a regular table with commas separating the horizontal cells.

2.1.7 JSON

JavaScript Object Notation (JSON) [11] is a common data format originally derived from the JavaScript language. Unlike CSV, JSON can handle nested data.

2.1.8 Apache Spark

Apache Spark is "a unified analytics engine for large-scale data processing" [12]. Apache Spark allows for relatively easy distributed computing and is heavily used within big data analytics to perform expensive tasks. Project MEP relies on Apache Spark for computations and data handling. A commonly used Apache Spark data structure is the DataFrame. A DataFrame is basically a table with rows being data entries and columns denoting information type. The point with the DataFrame is that it can support millions of rows and handle expensive analysis operations performed on the data it contains.

2.2 Tools and libraries

2.2.1 Atom

Atom [13] is a text editor developed by the company GitHub [17] that is free of charge and open-source. Atom is convenient to use to code websites.

2.2.2 Databricks

Databricks [14] is both a company and a computing platform. Databricks, the computing platform, is an online tool built on Apache Spark that constitues a user-friendly environment to perform big data tasks. Databricks consists of notebooks where code can written and executed with a workflow suitable for big data analysis. Several languages are supported, among them Scala. HTML websites can be displayed in a Databricks notebook if all the HTML, CSS, and JavaScript code is passed as an argument to the function `displayHTML(...)`. All Project MEP code is written in Databricks notebooks.

2.2.3 D3

Data-Driven Documents (D3) [15] is a JavaScript library to create interactive SVG visualizations for websites.

2.2.4 Node and npm

Node [16] is an environment that allows for using JavaScript to build web servers.

2.2.5 GitHub

GitHub [17] is an online hosting service that is very commonly used for code development. On GitHub, projects can be conveniently uploaded and shared. The part of the GitHub website where a specific project can be found is referred to as that project's GitHub repository or simply it's repository. In addition to the actual project, a repository usually includes a description of the project and how to get started with it.

2.3 Twitter terminology

- **Tweet:** An original post on Twitter, i.e. a very brief text written by a Twitter user that can include images and links. Retweets, quotes, and replies are also considered tweets.
- **Retweet:** A Tweet that is publicly shared by another Twitter user, with full credit given to the original Tweeter.
- **Quote:** A retweet with a text comment by the user that is retweeting.
- **Reply:** A response tweet to another tweet.
- **Follow** To connect to another Twitter user and receive notifications whenever the followed Twitter user tweets.
- **Follower count** How many followers a Twitter user has.
- **Retweet count** How many retweets a tweet has.
- **Retweet network** A graph with unique Twitter users as nodes and retweets as edges.
- **Timeline** A series of user posts on Twitter, i.e. tweets, retweets, quotes, or replies. A user timeline is a user's series of posts.
- **Username** A unique name assigned to every Twitter account. Not to be confused with the display name that can be changed by the user.
- **User ID** A unique number assigned to every Twitter account.

2.4 Tweet Transmission Tree

A Tweet Transmission Tree (TTT) is an Apache Spark DataFrame that has been developed within Project MEP. TTTs contain a collection of tweets. The usefulness of the TTT is that it is an organized, cleaned, and big data-compatible collection of Twitter's API output. Thus, a TTT can be thought of as Twitter data ready for analysis. Important for this project is that for each tweet in the TTT, there is information about

- the tweeter's username and user ID;
- the type of tweet, i.e. tweet, retweet, quote, or mention;
- which other user a tweet is interacting with, if any;
- the tweeter's followers count;
- the tweet's retweet count;
- the tweet's text content; and
- the time the tweet was posted.

2.5 Twitter retweet network

A way to model interactions on Twitter is to produce a retweet network. A retweet network is a graph with Twitter users as nodes and retweets as edges. A retweet network does not cover all Twitter interactions but as retweets are a profoundly central component of Twitter, it is a reasonably good interaction model. A benefit of a retweet network compared to other ways to model Twitter interactions is that a retweet network is well defined from Twitter's API data and relatively easy to produce.

3 Method

3.1 Determining visualization tool

There are different ways to produce interactive visualizations. Several commercial products exist, e.g. Tableau, as well as several open source libraries, e.g. Vegas and D3. A requirement for this project was that the interactive visualization should be integrable in a Databricks notebook. This requirement excluded most commercial solutions as they are usually separate programs and favored open source visualization libraries. The requirement to produce a interactive visualizations and not just static visualization further narrowed the search for a tool to produce interactive visualizations. Most interactive visualizations are made with JavaScript for websites and such JavaScript visualizations are possible to create in a Databricks notebook. Therefore, a JavaScript library was selected to build the interactive visualizations. After considering several JavaScript visualization libraries, D3 was chosen for this project. The main reason why D3 was chosen is D3's flexibility and extensive documentation. Furthermore, many JavaScript visualization libraries are built to produce charts rather than visualizations in general which D3 is, and simply producing charts would not suffice for this project.

3.2 Studies and preparatory development

In order to produce interactive visualizations with D3, a plethora of technical difficulties arose. Databricks, Scala, website development, and D3 had to be learned without any prior knowledge of those technologies. To acquire the knowledge required, the project was structured with studies during the first weeks. Several online tutorials on D3 and JavaScript were followed and studied. The Databricks environment rapidly proved itself dreadfully inconvenient to develop JavaScript visualizations in. A decision was made to develop the visualizations offline as websites using Atom and to later incorporate the visualizations in Databricks notebooks. This workflow seemed natural and convenient for visualization development with D3. However, more technical difficulties arose with this decision. Now, setting up a local website using a server had to be learned as D3 requires a server to function. Node was learned and used for the website server. Another

decision was made to develop a simple website where all the visualizations produced in this project can be explored and make the website accessible by uploading it to GitHub. Such a website would require a relatively small amount of extra work as the development was already conducted in a website format. When foundational knowledge of website development was acquired after two weeks or so, development and further D3 skills honing was a straightforward process.

After a couple of weeks, integration in Databricks notebooks started. Learning the basics of Scala and how to use the `displayHTML()` function did not prove too difficult. Now with the knowledge of how to create the interactive visualizations for this project, development of the final interactive visualizations could be started.

3.3 Data cleaning

Before developing the interactive visualizations for this project, what data to display needed to be decided. Through Project MEP, TTTs were readily available to use for the visualizations. To produce an interactive visualization that allows for exploration of a user timeline, no modification or computation of a TTT was needed. However, computations on TTTs were needed to produce interactive visualizations that allow for exploration and understanding of Twitter networks. A TTT only contains single tweets and no explicit statistics or explicit information about interactions among users. Programs to compute the retweet network in a TTT was developed in Scala in Databricks notebooks. The computed data was also made ready for download in a CSV format and made ready for usage in the `displayHTML()` function in a JSON format.

3.4 Developing the interactive visualizations

With all the data needed to finish the interactive visualization goals for this project now available, the development of the interactive visualizations could be started.

Firstly, the user timeline visualization was developed. The user timeline visualization layout was re-designed several times, which was the most time consuming aspect. The actual programming did not entail any particular problems beyond what could be expected or considered normal in a programming project.

Secondly, the network visualization was developed. The network visualization was significantly simpler than the user timeline visualization and did not take longer than a couple of days to develop.

Thirdly, the graph visualization was developed. Like the network visualization, the graph visualization was rather simple and did not take more than a couple of days to finish.

3.5 Website development

The website was developed simultaneously with the interactive visualizations. In addition to containing the visualizations, a simple main page with access to each visualization and some visualization descriptions were included in the website. When finished, the website was uploaded to GitHub. With no prior knowledge of GitHub, some time had to be spent learning how GitHub works. A simple but sufficient project description was written and added to the project's GitHub repository.

3.6 Databricks integration

With the visualizations and the website finished, the visualizations were created as Scala functions in separate Databricks notebooks. This required some minor modifications of the HTML, CSS, and JavaScript code used for the website and a compilation of the TTT computations. It was neither difficult nor that time consuming. In addition to the actual functions, a Databricks notebook with usage examples was created to facilitate understanding and use of the visualization functions.

4 Results

4.1 Interactive visualizations

4.1.1 User timeline visualization

An interactive visualization that allows for exploration of a Twitter user timeline was developed. A screenshot of this visualization is shown in figure 1.

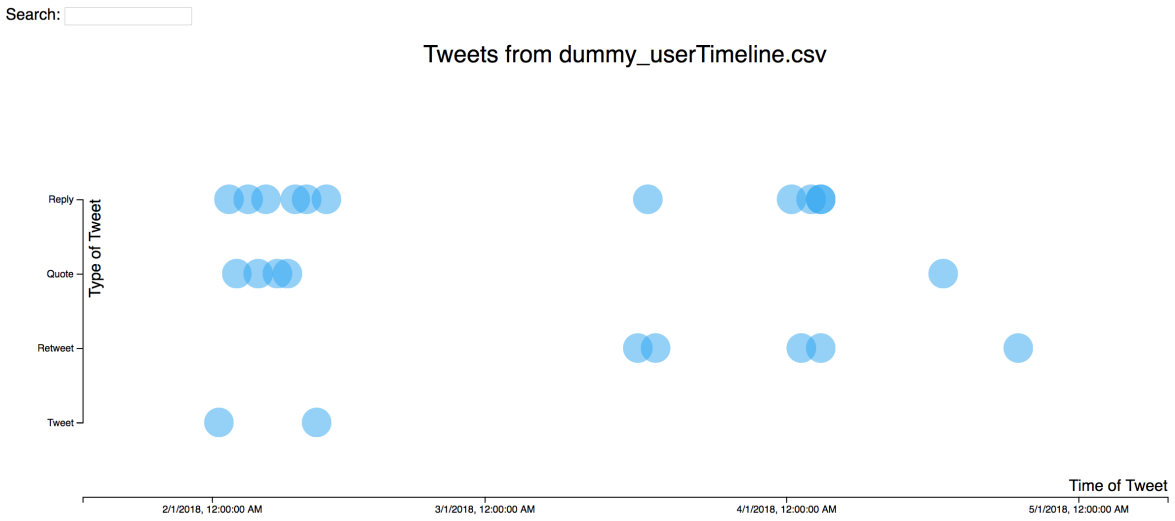

Figure 1: A screenshot of an interactive visualization that allows for exploration of a Twitter user timeline that was developed in this project.

In the user timeline visualization shown in figure 1, each circle represents a tweet. The x-axis shows the time a tweet was posted and the y-axis shows the tweet type of the tweet. The displayed time interval can be zoomed by scrolling. When hovering over a tweet circle, the tweet text content will appear in the upper left corner and the tweet circle that's hovered will become enlarged. Above the tweet text in the upper left corner, there is a search box that allows for searching all displayed tweets' text content. If the character sequence searched for appears in a tweet text, the corresponding tweet circle is enlarged and colored yellow. On the website but not in the Scala function, if a tweet circle is clicked, the actual tweet from Twitter's API will pop up on the right-hand side. The actual tweet can be clicked as a link to Twitter's website where the full tweet with replies can be viewed.

4.1.2 Network visualization

An interactive visualization that allows for exploration and understanding of a Twitter retweet network was developed. A screenshot of this visualization is shown in figure 2.

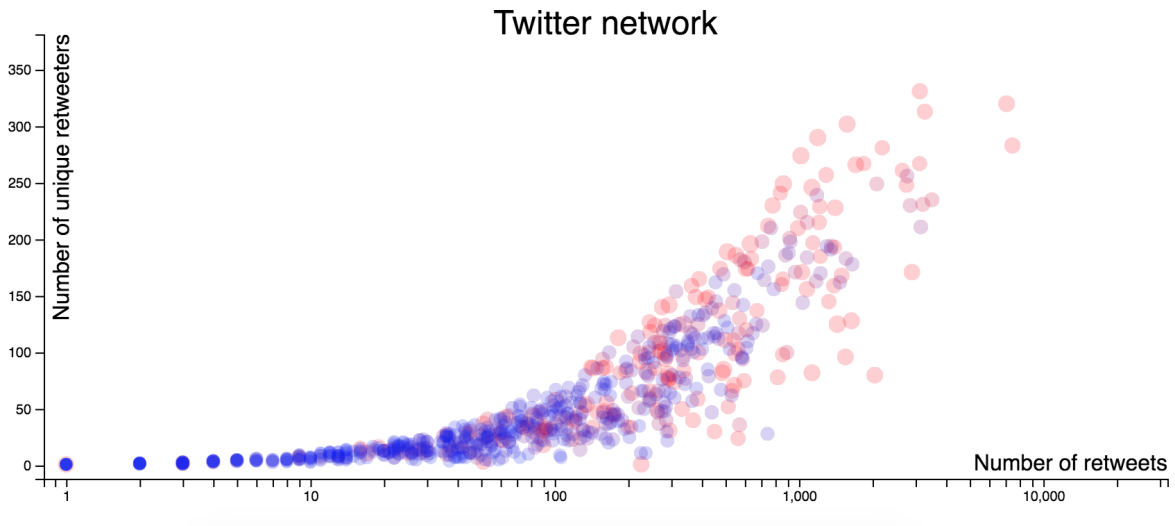

Figure 2: A screenshot of an interactive visualization that allows for exploration and understanding of a Twitter retweet network that was developed in this project.

In the retweet network visualization in figure 2, each circle represents a Twitter user, i.e. a unique Twitter account. The color of the user circle represents how many followers the user has, from blue (fewest) to red (most). The radius of the user circle also represents how many followers the user has, from small radius (fewest) to large radius (most). The x-axis shows the number of retweets of a user within the Twitter network, i.e. the number of times the user was retweeted by users in this specific Twitter network. The y-axis shows the number of unique retweeters within the Twitter network, i.e. the number of unique users in this specific Twitter network that retweeted a user. If a user circle is hovered, the user screen name will appear by the user circle and the user circle hovered will become enlarged. On the website but not in the Scala function, if a user circle is clicked, the up-to-date user timeline from Twitter's API will appear on the right-hand side. The user timeline can be clicked as a link to Twitter's website where the full user profile can be viewed.

4.1.3 Graph visualization

An interactive visualization graph that allows for exploration and understanding of a Twitter retweet network was developed. A screenshot of this visualization is shown in figure 3.

Figure 3: A screenshot of an interactive visualization graph that allows for exploration and understanding of a Twitter retweet network that was developed in this project.

In the retweet network graph visualization in figure 3, each circle represents a Twitter user, i.e. a unique Twitter account. The radius of the user circle represents how many retweets the user has within this specific Twitter network, from small radius (fewest retweets) to large radius (most retweets). If a user circle is hovered, the user screen name will appear by the user circle and the user circle hovered will become enlarged. The line thickness connecting user circles represents how many retweets the user circle tuple has in total, from small line thickness (fewest) to large line thickness (most). Retweet direction is not accounted for. On the website, connecting lines only appear if the user circle tuple has at least 15 retweets. In the Scala function, the tuple retweet cut off can be passed as a parameter but is set to 15 as default. On the website but not in the Scala function, if a user circle is clicked, the up-to-date user timeline from Twitter's API will appear on the right-hand side. The user timeline can be clicked as a link to Twitter's website where the full user profile can be viewed.

4.2 Website

A website where the three interactive visualizations can be accessed [18] was developed. A screenshot of the website's main page is shown in figure 4.

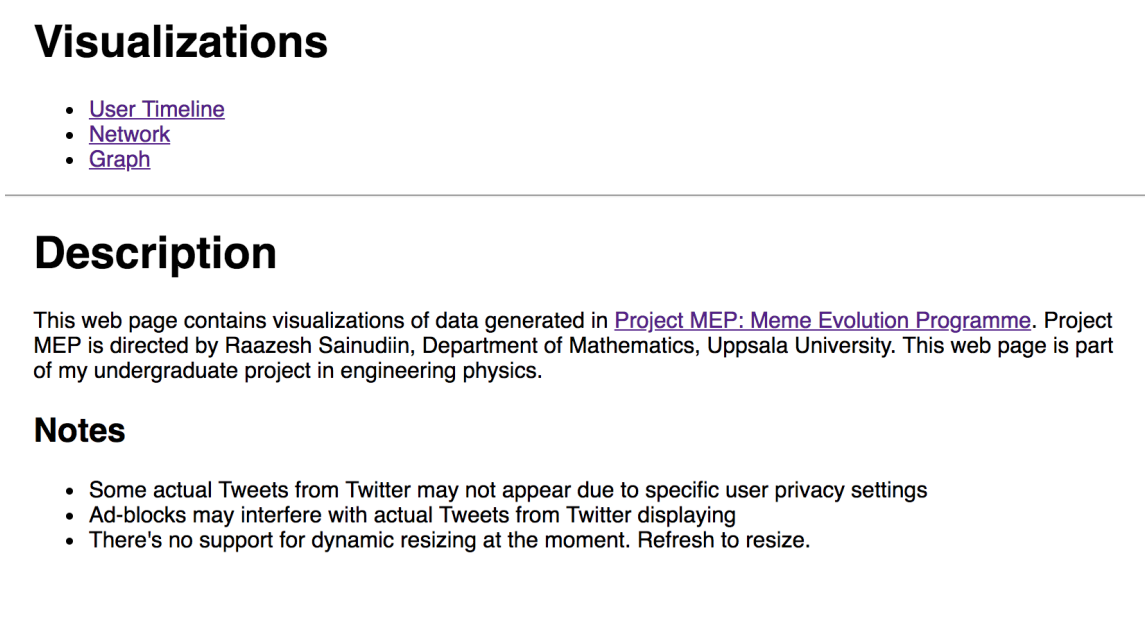

Figure 4: A screenshot of the main page of a website that allows for easy access to the three interactive visualizations developed in this project.

From the website's mainpage shown in figure 4, each of the three interactive visualizations developed in this project can be accessed by clicking on the link "User Timeline", "Network", or "Graph". When a link is clicked, the interactive visualization corresponding to that link is shown in the full browser window similar to what figure 1, 2, or 3 looks like.

The website was uploaded to GitHub and is readily available to download and use from the GitHub repository `/olofbjorck/twitterVisualizations`. However, the data needed for the interactive visualizations need to be collected and downloaded separately by the user that intends to use the interactive visualizations. The website requires Node to run. The details of how the data needs to be structured and how to get started can be found in the GitHub repository description.

4.3 Scala functions

For each interactive visualization, a Scala function in a Databricks notebook to produce the interactive visualization was developed. The functions have three arguments in common,

- a TTT;
- the display width; and
- the display height.

The TTT should contain data relevant to the specific interactive visualization. The width should be the desired visualization width in pixels and the height should be the desired visualization height in pixels.

4.3.1 User Timeline function

The user timeline visualization function displays the user timeline visualization and the function declaration is shown in listing 1.

Listing 1: A Scala function to produce an interactive visualization of a user timeline developed in this project.

```scala
import org.apache.spark.sql.DataFrame

/** Displays an interactive visualization of a Twitter timeline.
 *
 *  This function takes a DataFrame from TTTDFfunctions function tweetsDF2TTTDF()
 *  and displays it in an interactive visualization. The DataFrame content should
 *  be a Twitter user timeline.
 *
 *  Visualization description:
 *
 *  ...
 *
 *  @param TTTDF A DataFrame from TTTDFfunctions function tweetsDF2TTTDF() containing
 *               the data to be displayed.
 *  @param width The display width in pixels.
 *  @param height The display height in pixels.
 */
def visualizeTimeline(TTTDF: DataFrame, width: Int = 900, height: Int = 400): Unit = {

    // Display visualization
    displayHTML(...)
}
```

The TTT argument for the `visualizeTimeline` function should contain a specific user's tweets. The `visualizeTimeline` function performs no other operation on the TTT than displaying it.

4.3.2 Network function

The network visualization function displays the network visualization and the function declaration is shown in listing 2.

Listing 2: A Scala function to produce an interactive visualization of a Twitter network developed in this project.

```scala
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.functions._

/** Displays an interactive visualization of a Twitter network.
 *
 *  This function takes a DataFrame from TTTDFfunctions function tweetsDF2TTTDF()
 *  and displays it in an interactive visualization. The DataFrame content should
 *  be a collection of Twitter users.
 *
 *  Visualization description:
 *
 *  ...
 *
 *  @param TTTDF A DataFrame from TTTDFfunctions function tweetsDF2TTTDF() containing
 *               the data to be displayed.
 *  @param width The display width in pixels.
 *  @param height The display height in pixels.
 */
def visualizeNetwork(TTTDF: DataFrame, width: Int = 900, height: Int = 400): Unit = {

    // Computations
    ...

    // Display visualization
    displayHTML(...)
}
```

The TTT argument for the `visualizeNetwork` function should contain a selection of tweets. The `visualizeNetwork` function performs operations on the TTT to compute all retweet tuples and counts before displaying the

network visualization.

4.3.3 Graph function

The network visualization function displays the graph visualization and the function declaration is shown in listing 3.

Listing 3: A Scala function to produce an interactive visualization of a Twitter network developed in this project.

```
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.functions._

/** Displays an interactive visualization of a Twitter graph.
 *
 *  This function takes a DataFrame from TTTDFfunctions function tweetsDF2TTTDF()
 *  and displays it in an interactive visualization. The DataFrame content should
 *  be a collection of Twitter users.
 *
 *  Visualization description:
 *
 *  ...
 *
 *  @param TTTDF A DataFrame from TTTDFfunctions function tweetsDF2TTTDF() containing
 *               the data to be displayed.
 *  @param tupleWeightCutOff The display width in pixels.
 *  @param width The display width in pixels.
 *  @param height The display height in pixels.
 */
def visualizeGraph(TTTDF: DataFrame, tupleWeightCutOff: Int = 15, width: Int = 900, ...
    height: Int = 400): Unit = {

    // Computations
    ...

    // Display visualization
    displayHTML(...)
}
```

The TTT argument for the `visualizeGraph` function should contain a selection of tweets. The `visualizeGraph` function performs operations on the TTT to compute all retweet tuples and counts before displaying the graph visualization. In addition to the TTT, width, and height arguments, the `visualizeGraph` function takes a fourth argument, the `tupleWeightCutOff`. The `tupleWeightCutOff` argument should specify how many retweets a tuple requires to have a connection visualized between the tuple users.

5 Discussion

5.1 Usefulness

The interactive visualizations developed in this project can be somewhat useful within Project MEP. The visualizations themselves will probably not result directly in any research papers or discoveries but they can certainly be of use for exploratory data analysis. The user timeline visualization can be of use to quickly gain insight of a Twitter user which could be valuable if someone for example is exploring prominent politicians and what they have tweeted. The network visualization and the graph visualizations can be of use to understand a Twitter network which can be valuable to gain knowledge of Twitter interactions. For example, Project MEP is currently collecting large amounts of Twitter data from Swedish politicians for the upcoming Swedish general election. Visualizing that data will hopefully provide a useful overview of interaction within and between political parties.

5.2 Bugs

As this project have been conducted by an inexperienced JavaScript and D3 programmer, there is a definitive risk of there being bugs and loss of data in the visualizations. There is one prominent bug known; in the user timeline visualization, the first couple of data entries are not displayed. Specifically, the first six rows in the TTT are not displayed in the user timeline visualization for the Scala function, and the first row is not displayed in the website version. This bug has unsuccessfully been tried to fix and why the bug exists is completely unknown. This bug can be accounted for by adding dummy tweets in the beginning of the TTT.

Another bug, or rather a permission error, is that in the Databricks notebooks, the actual tweets from Twitters API do not appear. This is the case for all three of the interactive visualizations. This might be because of Databricks restricting connectivity to external API:s. There might be a solution to this problem but it has not been investigated in this project. However, the actual tweets from Twitters API seem to display and work as expected in the website version.

5.3 Future improvements

In addition to fixing the known bugs, there are two major improvement ideas. Firstly, for the network visualization and the graph visualization, it would be informative to color the users by some sort of group belonging, e.g. political party or country of origin. The D3 code for this is partly implemented but the group belonging data needs to be acquired in some way. Secondly, the network visualization could make use of an improved followers count visualization. Currently, each user circle's color and each user circle's radius represents the user's followers count. Both the color and the radius showing the same information variable is redundant and slightly awkward. The main reason why the network visualization was designed that way is because no satisfactory solution of visualizing the followers count with only the radius was discovered. Some users in the Twitter network might have a handful of followers and some might have over one hundred million. Even a heavily logarithmic radius scale has the effect of dwarfing most users or making some users enormous. The visualization could be improved by addressing this issue in some clever way.

There is a third, more obscure improvement idea compared to the previously stated improvement ideas. Because the interactive visualizations in this project could prove useful to researchers in the Humanities, it makes sense to let such researchers access them easily. Currently, non-trivial technical literacy is required to acquire data through Project MEP and to deploy the visualizations. This is a serious caveat for many researchers in the Humanities. An improvement would be to produce a user-friendly, non-technical way to access the visualizations created in this project. However, such an improvement might be an endeavor significantly more time consuming than this entire project.

6 Conclusion

6.1 Objective

The goal of this project was to produce interactive visualizations that allow for

- exploration of a user timeline;
- exploration of a Twitter network; and
- understanding of a Twitter network.

The goal is considered achieved. The user timeline visualization allows for exploration of a user timeline. The network visualization and the graph visualization together allows for exploration and understanding of a Twitter network.

6.2 Results

The results of this project is three interactive visualizations; a user timeline visualization, a network visualization, and a graph visualization. Each interactive visualization is accessible both through Scala functions in Databricks notebooks and in a website developed in this project. The website is accessible through GitHub

in the repository `/olofbjorck/twitterVisualizations`. The produced interactive visualizations are useful for exploratory data analysis of Project MEP datasets.

6.3 Improvements

There are known bugs that can be addressed to improve on this project. The user timeline visualization does not display the first data entries and the Scala visualization function do not allow connection to Twitters API. The network visualization and the graph visualization can be improved by coloring user circles by some sort of group belonging and the network visualization could be improved by visualizing users' followers counts in a better way than it is currently visualized.

References

[1] Twitter: Twitter,
retrieved 2018-05-15 from:
`https://twitter.com/`

[2] Raazesh Sainudiin: Project Meme Evolution Programme,
retrieved 2018-05-15 from:
`https://lamastex.github.io/scalable-data-science/sds/research/mep/`

[3] Joakim Johansson, *A Quantative Study of Social Media Echo Chambers*, 2018,
url retrieved 2018-05-17:
`http://uu.diva-portal.org/smash/get/diva2:1176971/FULLTEXT01.pdf`

[4] Wikipedia: Interactive visualization,
retrieved 2018-05-17 from:
`https://en.wikipedia.org/wiki/Interactive_visualization`

[5] École Polytechnique Fédérale Lausanne: The Scala Programming Language,
retrieved 2018-05-15 from:
`https://d3js.org/`

[6] W3: HTML,
retrieved 2018-05-15 from:
`https://www.w3.org/html/`

[7] W3: CSS,
retrieved 2018-05-15 from:
`https://www.w3.org/Style/CSS/`

[8] Wikipedia: JavaScript,
retrieved 2018-05-15 from:
`https://en.wikipedia.org/wiki/JavaScript`

[9] W3: SVG,
retrieved 2018-05-16 from:
`https://www.w3.org/Graphics/SVG/`

[10] Wikipedia: Comma-separated values,
retrieved 2018-05-15 from:
`https://en.wikipedia.org/wiki/Comma-separated_values`

[11] Wikipedia: JSON,
retrieved 2018-05-15 from:
`https://en.wikipedia.org/wiki/JavaScript`

[12] Apache: Spark,
retrieved 2018-05-15 from:
https://spark.apache.org/

[13] GitHub: Atom,
retrieved 2018-05-15 from:
https://atom.io/

[14] Databricks: Databricks,
retrieved 2018-05-15 from:
https://databricks.com/

[15] Mike Bostock: Data-Driven Documents,
retrieved 2018-05-15 from:
https://d3js.org/

[16] Node.js foundation: Node,
retrieved 2018-05-15 from:
https://nodejs.org/en/

[17] GitHub: GitHub,
retrieved 2018-05-15 from:
https://github.com/

[18] Olof Björck: twitterVisualizations GitHub repository,
retrieved 2018-05-21 from:
https://github.com/olofbjorck/twitterVisualizations