Tracking Interconnected Twitter Links

Using Graph Database Neo4j

Lindiwe Mncwabe - 1118055
Clifford Ralikhwatha - 0412610
Thomas Johannsen - 721988

2017

Contents

List of Figures

1 Introduction

1.1 Overview

This purpose of this project is to graph Twitter data so that hidden trends and patterns may be revealed.

1.2 Neo4j

Neo4j is the leading graphic database management system, that distinguishes itself from other systems through its ease of use and its speed. Its was initially launched in 2010. Its fundamental design is to store data as nodes, edges and attributes. Nodes are connected by edges, both can have any number of attributes.

1.3 Requirements

To run the code as desired the following programs or packages are required Neo4j v3.2.4
Python v2.7.13
PyCharm 2017.2.3
TextBlob v0.13.0
Tweepy v3.6.0
py2neo v3.1.2
CypherQueries(*GitHub*)
Twitterneofinal(*GitHub*)

1.4 Running the code

The specifics of running the code as desired can be found on `GitHub in the ReadMe` document

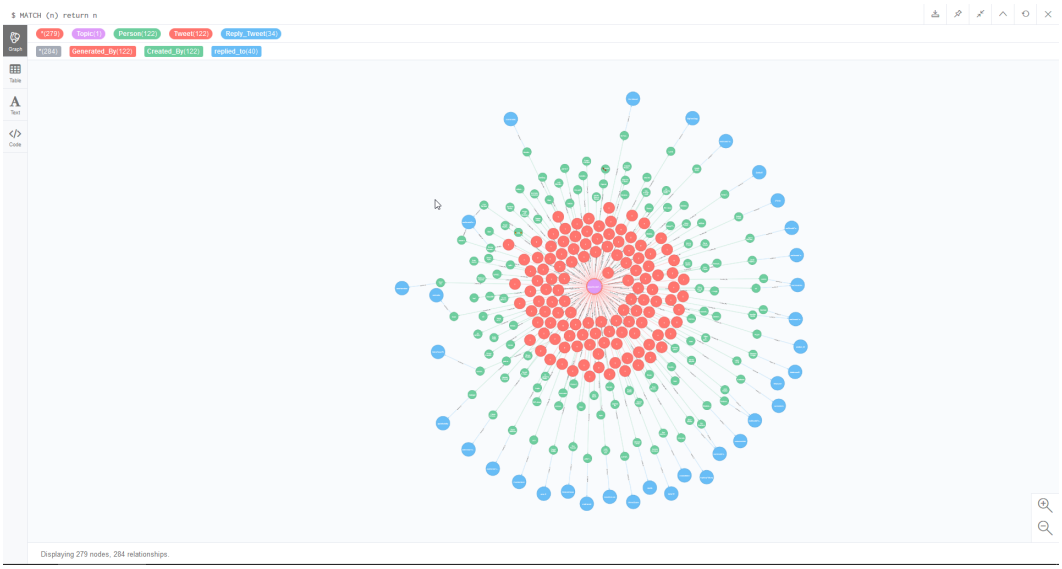

Figure 1: Neo4j browser look

2 Description

We import public data from twitter and store it in a Neo4j database. This database is graphically represented through the Neo4j browser, which shows nodes and connections, in different colours and sizes to highlight and differentiate various things. Filters can be applied to customise or limit what the graph shows. For example only tweets with a certain phrase or hashtag, "#neo4j", can be shown. This can give an idea about the age, location or gender of people tweeting about Neo4j.

2.1 Design

Figure 1 shows a demo of the final look.
Green nodes are people on twitter, labelled by their twitter handle, they share an edge labelled "created by" to red nodes, labelled "T", with anything they tweet. Should anyone retweet this tweet, then a "RT" node is created in purple, sharing "created by" edges with the original tweet and the retweeter. A blue node is created in a similar fashion, when someone replies to or mentions someone else on Twitter. The purple node in the center is the topic we chose to filter the tweets through. We thought "Trump" should always be trending, although in different volumes, based on the time of day. The central topic shares an edge with all shows tweet, labelled "generated by".
Every time the python script is run it increases the database size, the figure below was created after running the code just once. Twitter limits what we can access, running the script again an hour later brings in a similar quantity of data.

Figure 2 shows the data process, from Twitter to the final front-end interface.

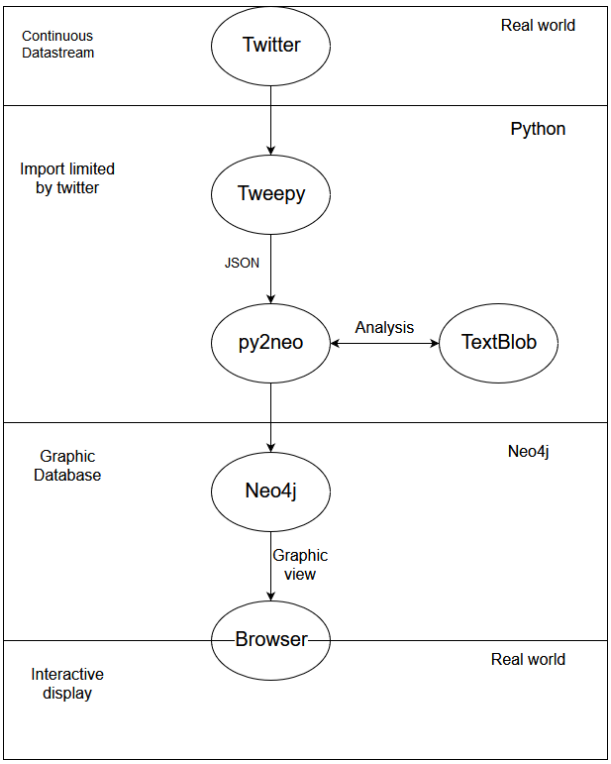

Figure 2: Data process

The Tweepy API collects data from the continuous flow of information that is Twitter, and converts it to a JSON string. This string is then processed to Neo4j data and analysed by TextBlob. The data is then passed to Neo4j, where it is stored and displayed as a graph. The graph can be manipulated and filtered through the Neo4j browser

Figure 3 shows the basic idea behind the set up of the database.

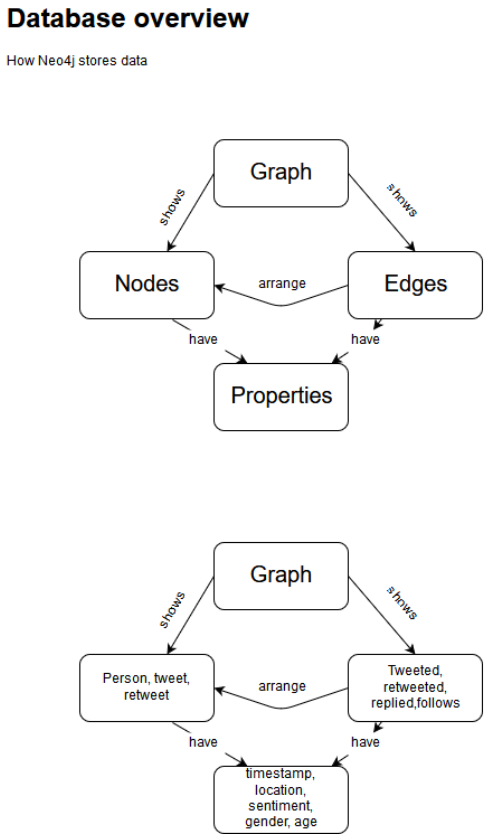

Figure 3: Neo4j database set up

Our nodes are people, tweets, retweets, replies and the central topic, they have various properties, such as age, gender, sex (where appropriate), time created, location. The edges "generated by", "created by" and "replied to" link the nodes. Edges can have properties too, properties not only provide additional information, that can be seen by clicking or hovering over a node/edge, they also can be used as filters to search/limit the data.

To emphasize how Neo4j simplifies the data being worked with to a graph Figure 4 shows the JSON string, the raw information, and the equivalent graph.

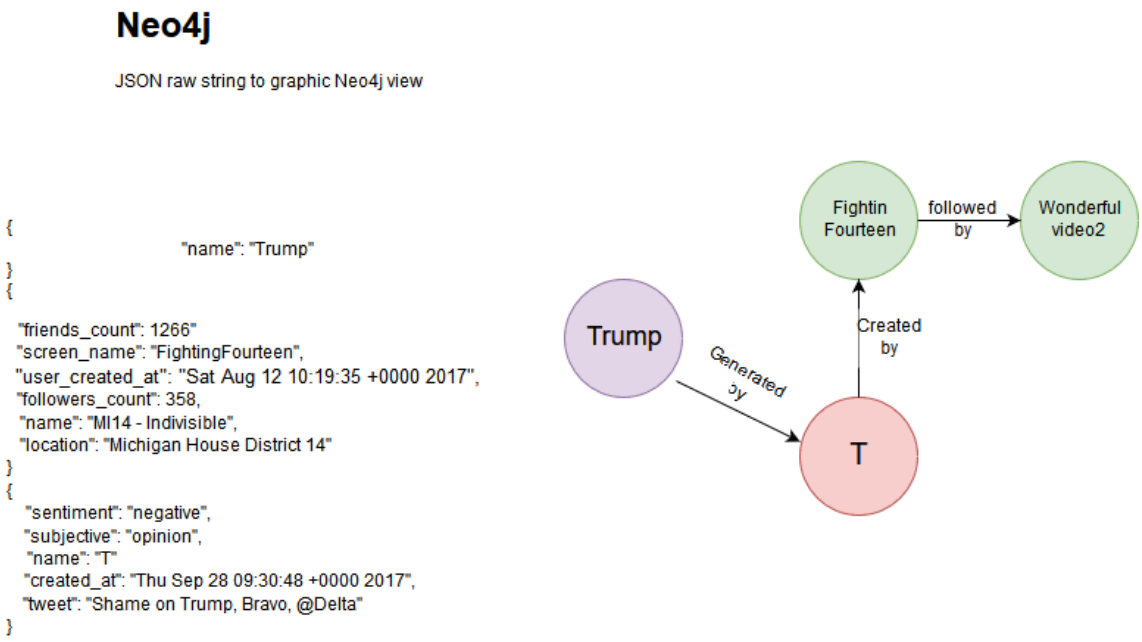

Figure 4: Raw data vs graph

The graph is a better presentation, that offers a better overview and logical structure of interconnections. This is mainly because it is easy to look at a graph showing the connection, than trying to imagine it for oneself.

Additionally the graph shows a lot less information at the first glance, data not relevant to the connection network aren't shown in the overview, but can easily be accessed by selecting a node/edge. The viewer can choose what information he wants, and doesn't need to sift through lines and lines of text.

The JSON for "Wonderfulvideo2" isn't shown, as it is very similar to that of "FightinFourteen", which in contained in the second set of bracket. Details such as friends, followers and time aren't shown on the graph, but hovering over an element, or selecting it, will reveal these and other properties.
We discussed including the location of where the tweet was made, however this is problematic for several reasons

- Not all users share their location with twitter
- Location isn't always accurate
- Different providers share locations differently (3 locations from our testing include, "#GameofThrones", "Basin, WY, United States", "Kenya")

In the end we left the location as a property of the nodes, which means it can be used as a filter. However further analysis of the location into a uniformed method would have to be applied for this information to be fully utilised.

Using a text analysis tool (TextBlob) we can look at the content of a tweet to see whether it's an opinion. We can also look at the sentiment of the content to determine whether a tweet is positive or negative. These are stored as properties which again can be used as filters, showing only tweets that are positive opinions, or no opinions at all...

2.2 Back-end

The back-end of the project imports data from Twitter to python as a JSON stream. The JSON is then analysed and converted to Neo4j data. In the process Nodes and edges are created and properties are assigned. External analysis such as sentiment are done before the conversion to Neo4j, and passed along as properties. We don't have a server to host the Neo4j data, which means that when a different machine runs the code they will collect separate data from twitter, based on what is trending at the time. This can result in differences based on when and how often Twitter data is collected, however by using a topic like "trump" we hope to always have a fair amount of hits.

2.3 Front-end

The front end interface is the Neo4j browser. Unfortunately we don't have a server to host this, so system settings can't be saved, such as the graphic template for colour, size and the like. This also means that changes to the data being viewed are done as Cypher queries. The browser itself allows for movement of nodes, zooming and inspection of elements as well as style changes, all of which can be done from the tool bar around the view panel. Editing the edges, requires a node to be selected and the "Expand child relationships" to be toggled, around the node. Properties of individual nodes can been viewed by hovering over a node, or selecting it and viewing either the table or text views, from the tool bar on the left.

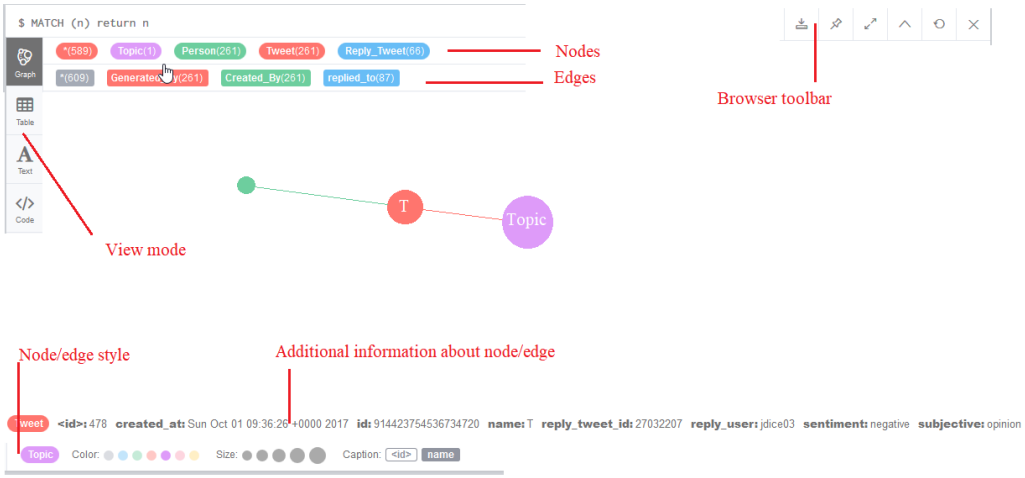

Figure 5: Browser interface

An overview of the nodes and connections in the database, differentiated by colour. Properties of nodes can be seen by clicking on a node. Node/edge style can be modifies by selecting the type of node/edge at the top of the graph and then styling them at the bottom.

View mode offers different ways to see the data, or code, "Table" and "text" show more text heavy versions of databases while "code" shows how the code runs to return results for a query. All view modes offer the same information, it comes down to personal preference which is preferred as certain data is more apparent in different view modes.

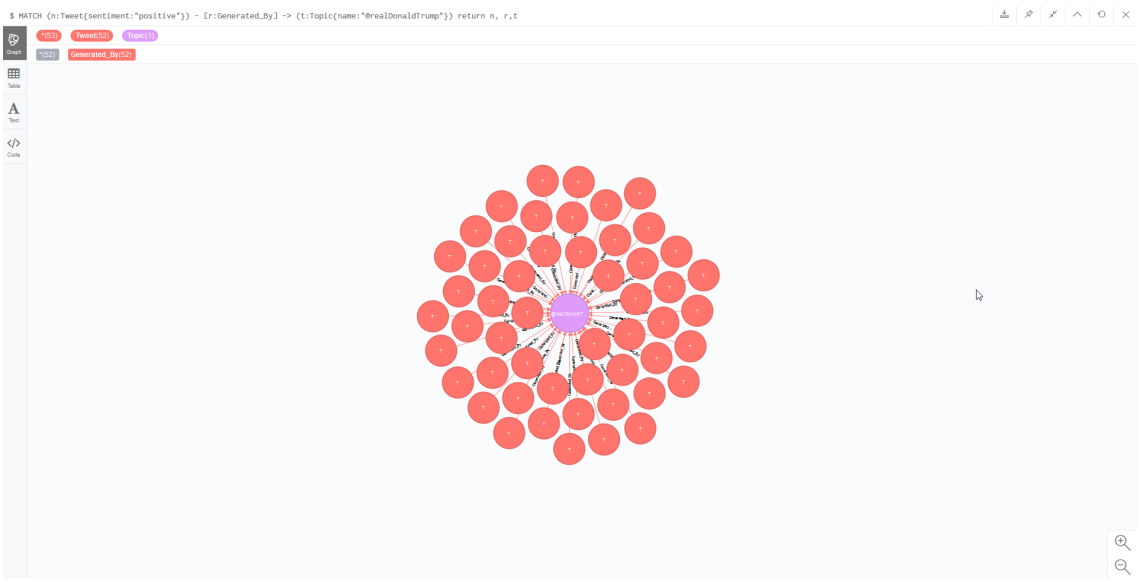

Figure 6: Positive tweets

$Running("MATCH(n:Tweetsentiment:"positive") - [r:Generate\_By] - > (t:Topic\{name:"@realDonaldTrump"\})returnn, r, t")$ This could be combined with a second query each tweet to its tweeter, or to see other activity involving these tweets should that be desired.

2.4 Application

The Neo4j database system is nice to look at and fun to play around with, beyond that it has real world applications. These applications are manifold and extend beyond just Twitter data. As we looked at Twitter data, lets look at how this data can analysed. We can track frequency of which people tweet about a topic based on the time of day. We can look at what a certain age/race/gender typically tweets, what sentiment it has, and how many retweets it gets. A company could see how many and what kind of person is tweeting about their product. The negative sentiment filter can be applied to only show people complaining about the product, to address community issues.

The hope is that the above queries and other similar ones will reveal hidden patters, once such pattern have been identified algorithms and further filters can be created to analyse these patterns are try find more. One of these hidden patterns was profiling, which a lot of online websites use when recommending you anything or showing you adverts.

2.5 Looking forward

The first improvement we would like to make would be to host the Neo4j database on a server, so that visual designs and templates can be stored for a uniform look. This also means that there would be a uniform database for users to use, and not a custom generated one based on the time of the execution. The next would be a script, either through a website or a program to automate the initial set up in Python, so that the user just needs to select his preferences or enter a topic into a search field and the script does the rest and displays the results in the browser. This could also eliminate having to run Cypher queries manually, as the script would take care of that, the user just needs to select his preferences and hit update.

3 Members

Back-end (Clifford)
Front-end (Thomas)
Documentation, model design (Lindiwe)
Testing (All)