University of Nebraska Omaha

University of Nebraska at Omaha
DigitalCommons@UNO

Interdisciplinary Informatics Faculty Proceedings & Presentations

School of Interdisciplinary Informatics

2015

FBWatch: Extracting, Analyzing and Visualizing Public Facebook Profiles

Lukas Brückner
Kyto GmbH

Simon Caton
National College of Ireland

Margeret A. Hall
University of Nebraska at Omaha, mahall@unomaha.edu

Follow this and additional works at: https://digitalcommons.unomaha.edu/interdiscipinformaticsfacproc


Part of the Communication Technology and New Media Commons, and the Social Media Commons

Recommended Citation

Brückner, Lukas; Caton, Simon; and Hall, Margeret A., "FBWatch: Extracting, Analyzing and Visualizing Public Facebook Profiles" (2015). *Interdisciplinary Informatics Faculty Proceedings & Presentations*. 8.
https://digitalcommons.unomaha.edu/interdiscipinformaticsfacproc/8

This Conference Proceeding is brought to you for free and open access by the School of Interdisciplinary Informatics at DigitalCommons@UNO. It has been accepted for inclusion in Interdisciplinary Informatics Faculty Proceedings & Presentations by an authorized administrator of DigitalCommons@UNO. For more information, please contact unodigitalcommons@unomaha.edu.

Dr. C.C. and Mabel L. CRISS LIBRARY

FBWatch: Extracting, Analyzing and Visualizing Public Facebook Profiles

Lukas Brückner, lukas@lukas-brueckner.de, Kyto GmbH

Simon Caton, Simon.Caton@ncirl.ie, National College of Ireland

Margeret Hall, hall@kit.edu, Karlsruhe Service Research Institute

An ever-increasing volume of social media data facilitates studies into behavior patterns, consumption habits, and B2B exchanges, so called Big Data. Whilst many tools exist for platforms such as Twitter, there is a noticeable absence of tools for Facebook-based studies that are both scalable and accessible to social scientists. In this paper, we present FBWatch, an open source web application providing the core functionality to fetch public Facebook profiles en masse in their entirety and analyse relationships between profiles both online and offline. We argue that FBWatch is a robust interface for social researchers and business analysts to identify analyze and visualize relationships, discourse and interactions between public Facebook entities and their audiences.

1 Big Data Challenges in the Social Sciences

The vision of a Social Observatory is a low latency method for the observation and measurement of social indicators. It is a computer-mediated research method at the intersection of computer science and the social sciences. The term Social Observatory is used in its original context (Lasswell 1967; Hackenberg 1970); the framework is the archetypal formalization of interdisciplinary approaches in computational social science. The essence of a Social Observatory is characterized by (Lasswell 1967) as follows:

"The computer revolution has suddenly removed age-old limitations on the processing of information [...] But the social sciences are data starved [...] One reason for it is reluctance to commit funds to long-term projects; another [...] is the hope for achieving quick success by 'new theoretical breakthroughs' [...] It is as though we were astronomers who were supposed to draw celestial designs and to neglect our telescopes. The social sciences have been denied social observatories and told to get on with dreams"

This is also in line with the approach of the American National Science Foundation's call for a network of Social Observatories:

"Needed is a new national framework, or platform, for social, behavioral and economic research that is both scalable and flexible; that permits new questions to be addressed; that allows for rapid response and adaptation to local shocks […]; and that facilitates understanding local manifestations of national phenomena such as economic volatility."

Today, the notion of a Social Observatory lends itself towards social media platforms, as digital mediators of social exchange, discourse and representation. This, as demonstrated by the COSMOS project (Burnap et al. 2014), becomes especially valuable when combined with government data streams. However, empowering social scientists to access data from social media platforms (even in the singular) is non-trivial.

Figure 1 illustrates a general architecture of a modern Social Observatory entailing three processes; namely 1) Data Acquisition; 2) Data Analysis; and 3) Interpretation. Whilst it is apparent that a Social Observatory captures multiple sources of data, currently few scientific papers or services report this ability in a way easily replicable by social scientists (Cioffi-Revilla 2014). This is despite prevalent availability of Application Programming Interfaces (APIs), and an almost endless supply of papers and studies that focus on specific platforms (Russell 2013).

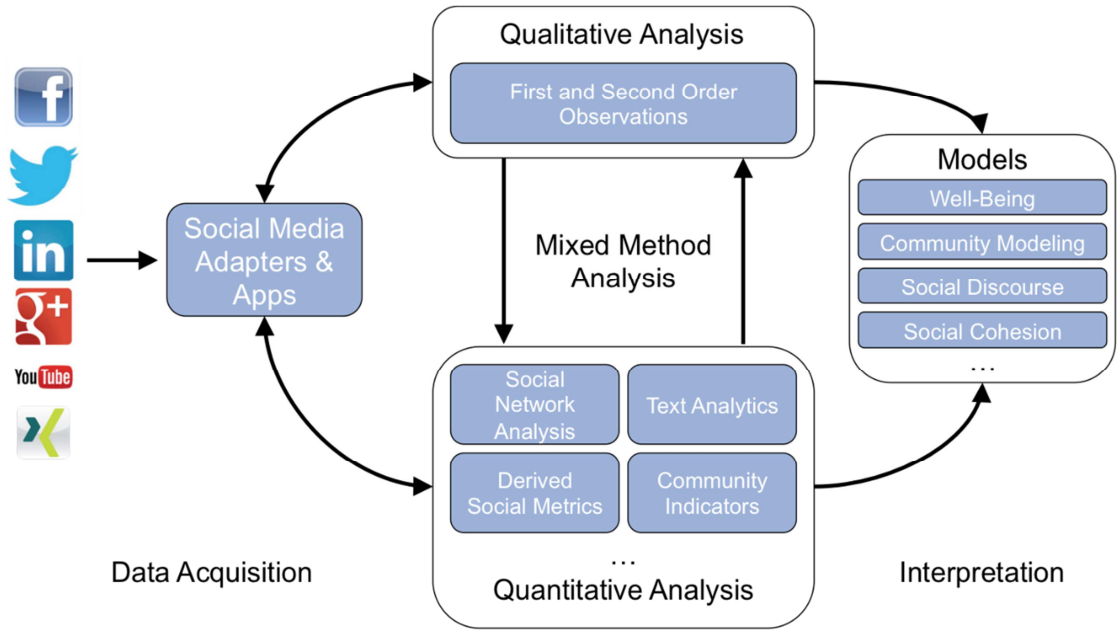

Figure 1. A General Architecture for a Social Observatory

Data Acquisition is well supported by most social media platforms via REST or streaming APIs, which are underpinned by lightweight data interchange formats like JSON. User authentication and access authorization is handled by technologies such as OAuth. There are also an ever-increasing number of software libraries available, reducing the implementation effort to extract data.

The challenges instead lie in data *volume*, *velocity*, *and variety*, access rights, and cross-platform differences in curating data. The big data aspects of social media data are well known: producing 2,200 Tweets (at around 58kb each) per second, Twitter is a clear demonstrator of data volume and velocity. Variety is best shown using a Facebook post as an example: version 1 of Facebook's Graph API contained at least 15 categories for a user post and this discounts other social actions like tagging, commenting, poking etc., as well as the diverse content range of a Facebook user's profile. Lastly, the method of data curation is not without its ambivalence. Twitter data

curation tends to be proactive; by accessing future Tweets that fulfil a specific set of user-driven attributes (e.g., hashtags or geolocation). Facebook is retrospective; given a Facebook entity (e.g. a person, or page) access their posts, profile, likes etc. From the perspective of analyzing social data, this subtle difference significantly alters the effort and planning needed to curate a data set (González-Bailón, Wang, Rivero, & Borge-Holthoefer, 2014). The technical challenges also differ significantly from receiving a continuous stream of data (i.e., tweets) vs. Facebook's paginated results. The latter incites large numbers of API calls, which are not limitless. On a side note, the validity period of an access token is also not infinite and must be refreshed periodically.

(**Mixed Method**) **Analysis** as illustrated in Figure 1, is inherently iterative and interdisciplinary. Foreseeable is repeated interaction with the social media adapters and apps. Whilst approaches from computer science and computational social science are becoming more prevalent, the question of research methodology is often a poignant discussion point and challenge that cannot be overlooked. Computer scientists and social scientists speak very different languages. Therefore, the realization of a Social Observatory needs to accommodate a vast array of (interdisciplinary) methodological approaches.

Irrespective of methodology, an important feature of a Social Observatory is the ability to view a community at a variety of resolutions; starting from an individual micro layer, and progressively zooming out via ego-centric networks, social groups, communities, and demographic (sub) groups, up to the macro layer: community. This ability is of significant importance for understanding a community as a whole; different granularities present differentiated views of the setting. Interpretation is hence domain specific in nature, and should be decided according to the proposed research questions. The architecture supports both inductive and deductive research.

Necessary to address at this point are the ethical boundaries of an unobtrusive approach of Big Data analyses of social data. Both Twitter and Facebook have terms and conditions allowing for the anonymized assessment of data which the use has indicated to be public. Specifically Facebook has argued that this is tantamount to informed consent, and this is a common position across social media platforms. This study agrees that when information is placed in public fora and domains, it is subject to public review. This is in line with the ethical guidelines of (Markham & Buchanan, 2012). In the case of obtrusive design (i.e., greedy apps), informed consent must continue to be in place as the standards of human subject research demand. A further ethical (and security) concern is that the provide architecture can also be used irresponsibly. In the case of public-facing data, this is of a lesser concern. Obtrusively-designed architectures still require user consent (e.g., downloading an app), as such research works are neither the work of hacking nor 'Trojan horses,' thus guaranteeing a moderately informed subject base.

1.1 Implementation: a Facebook Social Observatory Adapter

The first step towards a Social Observatory focuses on a Facebook social adapter for several reasons. Firstly, Facebook lends itself to the case study, especially due to the large number of "open" Facebook entities; where Facebook pages are a prime example. Secondly, when extracting data from Facebook, the researcher receives

near complete datasets. Finally, there is lack of general-purpose Facebook data acquisition tools available. Those that are available tend to rely either on crawling techniques, which cannot fully acquire paginated Facebook data, or data extraction via the Graph API that typically focus on the logged-in user or do not return data in full. Whilst such approaches are useful, especially in classroom settings, they do not provide mechanisms to curate research worthy datasets. This chapter presents a general and extensible Facebook data acquisition and analysis tool: FBWatch.

The objective is simple: an interface-based tool allowing social as well as computational scientists to access complete Facebook profiles irrespective of programming ability or data size, as no such tool is available. In extracting data from Facebook, the researcher first needs to define what is accessed: an entity that has a unique Facebook identifier. FBWatch is implemented such that it can access any Facebook entity that is public, or for which it has received user permissions.

FBWatch is implemented using the Ruby on Rails framework, and consists of five top-level components and modules: 1) Sync is the module responsible for fetching data from Facebook. It executes Graph API calls, converts graph data to the internal data structures and stores it in the database.b2) Metrics are the analysis components of FBWatch and responsible for analyzing fetched data. They contain parameters used for case studies and data structures for storing results. A metric can therefore be any result of an analysis (see Section 4). 3) Tasks are an abstraction for running Sync and Metric jobs as background processes. 4) A relational database for storing Facebook resource data, and running more complex queries regarding connections between Facebook entities. Any SQL-Server can be used provided that it supports UTF-8 encoding, as this is needed for handling foreign languages. MySQL and PostgreSQL both proved adequate. 5) A web front-end as an access point and controller for FBWatch. Here the user can request the retrieval of new Facebook entities, refresh previously fetched entities, group entities together for comparative analysis, execute metric calculations, visualize metrics as well as the social network of individual or grouped entities, and download datasets for use in third party analysis tools (see Section 3).

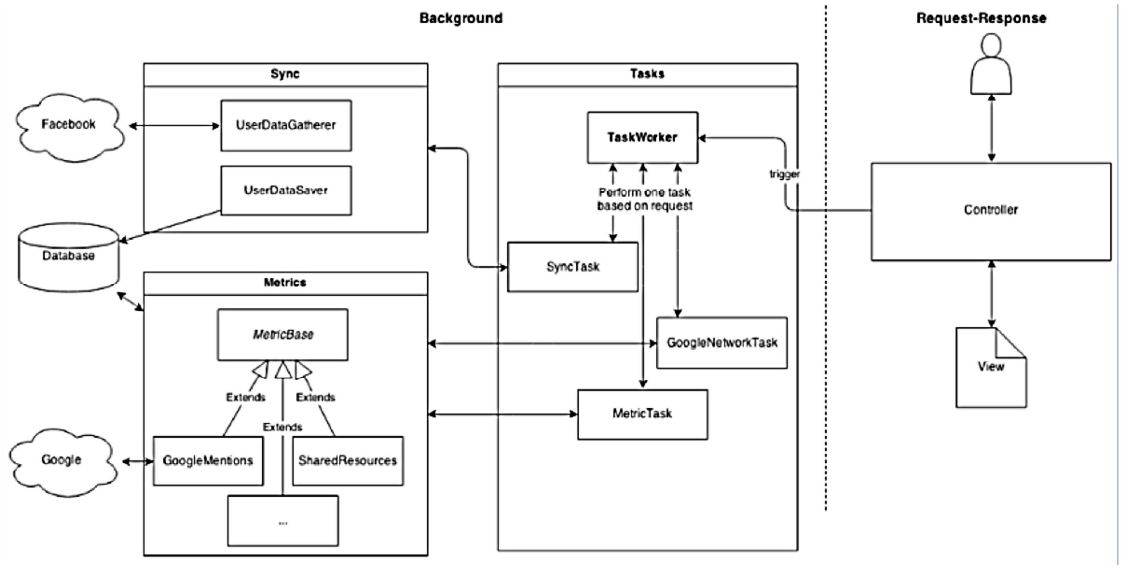

Figure 2. Workflow illustrating the steps to acquire, analyses, and interpret Facebook

Figure 2 shows the architecture of FBWatch, and highlights a typical request involving either the data fetching, or the metrics calculation. Upon a request, the controller triggers a background worker class and returns an appropriate view to the user who is notified that a task was started. The worker then performs one of two tasks, depending on whether Facebook data is to be retrieved, or retrieved data is to be analyzed.

The first step in the process flow the user providing the Facebook URL of one or more entities of interest, which are parsed for their username or Facebook ID. To synchronize the data of Facebook resources, a background sync task is started by FBWatch. The user can check the status and progress of the task, as required. Depending on the size and number of entities, synchronization can take several hours, and can also encounter several errors that need to be handled manually. Once synchronization has successfully completed, this will be visible and the user informed of how many feed entries have been retrieved. If errors were encountered that could not be handled this will also be displayed.

To access data, Koala, a lightweight and flexible Ruby library for Facebook, is used. It provides a simple user interface to the Graph API and the Facebook Query Language. As the Graph API returns the data in JSON format, Koala automatically parses the resulting string and converts it into the appropriate data structure using Arrays and Hashes and aligns the primitive data types into Ruby's data types. Furthermore, the library supports the use of the OAuth protocol to authenticate within Facebook through the use of the OmniAuth Ruby library. A valid, i.e. Facebook authenticated, instance of Koala is generated on a per-session basis and stored in the session context. At this time this is also the only real authentication the application performs directly. To mitigate exposing all data fetched by FBWatch, HTTP authentication is enforced on the server.

Synchronizing a Facebook resource is done in a two-step process. First, any basic information of that resource is pulled by calling the Graph API link facebook-id. Basic information contains the information visible at the top

of a Facebook page and in the about section, like first and last names, website, the number of likes etc. Second, the actual feed data is retrieved.

This is not trivial. First of all, not all data will and can be received at once, as Facebook limits the number of results per query; 25 per default. Increasing this limit drastically reduces the number of Graph API calls, and thus, speeds up the data gathering process. By default FBWatch uses a limit of 900, increasing speed and managing scalability. Facebook also only returns a subset of the comments and likes of a feed item; four by default. The resulting data contains a paging feature, similar to the one of the feed itself in a single feed item. Comments as well as like arrays have to be fetched using multiple API calls, dramatically increasing runtime. The UserDataGatherer module automatically navigates the paging system until it receives an empty data array. FBWatch also stores the link representing the first response from Facebook. This allows FBWatch to easily update a resource at some point in the future. If, however, a problem occurs, the last feed query is stored to enable the future continuation of a sync task.

The second part of the Sync module stores fetched data via the UserDataSaver. Aside from transforming Facebook JSON into internal data models, data entry needs to be optimized such that it scales. In order to decrease runtime, multiple INSERT and UPDATE statements are grouped into transactions. However, not all statements can be executed in one transaction due to interdependencies between data models. Thus, saving the data in the correct order is important. In order to take into account all possible dependencies, four transactions are used: 1) resources and their basic data are updated as well as all new Facebook entities that posted or interacted on the feed at the root level. 2) Feed entries. 3) Resources which interacted at a lower level, i.e. with a comment, like or tag. 4) The comments, likes and tags.

Once an entity has been fetched, it can at any time be resynchronized to retrieve any new feed items and their properties or continue to fetch all historic data if the synchronization was not successfully completed before. If a resource is no longer available on Facebook or no longer relevant for the analysis it also can be disabled or removed. Apart from the ability to traverse Facebook data automatically using the provided paging mechanism, the other main feature of the UserDataGatherer is error handling. The Facebook API is not reliable all the time, and is badly documented. Therefore, flexible error handling is required. The most pertinent hurdle is a limit to the amount of calls a single application can execute for a given access token in a certain time frame from the same IP address. While it is not officially documented, as per Facebook, apps tend to be limited to 600 calls every 10 minutes. For large resources, this limit is hit multiple times. FBWatch handles this by pausing the sync task, and retrying periodically (every five minutes) to resume it. This can require up to 30 minutes. FBWatch also handles when a resource cannot be queried, be it that it was deleted or disabled, when a username has been changed, and other miscellaneous errors.

1.2 Data Model

The data models representing social network data is loosely based on the Facebook Graph API format. A resource model corresponds to one Facebook entity but also constitutes the most important object in FBWatch. All overlapping properties of the different types of Facebook resources are saved in this data model: the free text name, the unique Facebook ID, the unique username and the full link to the resource on the Facebook system. Additional data relevant for the application is saved in this data model as well: a flag indicating whether or not a resource is active, i.e. if it should be synchronized, and the date of the last synchronization.

Other information returned by Facebook differs greatly for different entity types and is thus stored as an array of key-value pairs. Here, information such as the number of likes for pages, a website URL or the first and last names of real users, their gender and email address is represented. Furthermore, configuration data of the application is stored: information of the last synchronization so that it can be resumed more easily and no duplicates are retrieved. The value of stores the URL of the first link of the paging feature of the first feed page, i.e. where at the moment of synchronization newer data would be available. A property is called ‘last link’ stores the link to the last feed page unsuccessfully queried if an error occurred.

The core data structure is the feed (or timeline); a set of feed items. A feed item is modeled such that any type of textual activity can be represented, i.e. posts, comments and stories. Obviously, stories play an important role in user feeds. Note, however, that stories often appear right next to the actual activity, especially for comments; therefore, the content will be duplicated without care. So as to not lose too much information when handling different types of feed entries, a few additional properties are needed to the standard Facebook set. In order to simplify the data model differences in the available post types are mostly ignored. Post types are links, photos, statuses, comments, videos, swfs (flash objects) and check-ins as well as the corresponding stories. After analyzing the properties of these entries, the following attributes were selected: the unique facebook ID, timestamps representing when the entry was created and when it was last updated, the originator of the entry, optionally also the receiver of the entry and the comment and like count if present.

The originator and receiver are represented as separate resources, hence, only their unique IDs are stored here. The count of comments and likes are taken from the comments and likes properties of the Facebook format if present. A normal post has an attribute message which holds the text the user posted. A story, however, does not have a message, but rather a story property. The different sub-types of a post additionally have attributes containing the link, photo URL, etc. Each of these properties are mapped onto a single property. In order to distinguish between different types of feed items this property can be any of message, story or comment. The attribute then holds either story or comment for these two data types and the concrete post type for messages. A foreign key to the resource which this feed item belongs to, i.e. on which timeline it is posted. Last, to link comments to their respective post, a parent property is included, which is null for top-level posts.

1.3 Summary

The developed artifact demonstrated a first prototype of forming a general service that is capable of facilitating Big Data analyses based on Facebook data. The resulting software was designed to be modular enough to be extended in many different possible ways in order to support a multitude of research questions. As an endeavor like this is a large project only a first foundation was implemented. Nevertheless, as a first exploratory work in that direction the feasibility of a larger service was demonstrated. The aim of targeting software towards non-computer scientists is met for the main workflow. For this main workflow the other usability requirement of response times of less than ten seconds is met. Clearing data or loading the deep details of a resource can take more than ten seconds. For future applications to be performed on a different set of resources, the application provides a simple workflow without the need to adjust any source code. Modifying the scoring or adding new metrics requires programming knowledge, but is feasible.

In order to facilitate different analyses, the metric system was modularly defined. By providing a general base class where all specific metric classes can register themselves, it can be easily extended. Should external systems be required to perform additional analyses, the fetched data can be exported into a JSON format and put to other software. The structure of the JSON format was designed to be close to the one Facebook provides itself. Since not all returned data is saved and some parts are stored differently, the JSON feed of Facebook and FBWatch are not a one-to-one match. Only small differences exist, though, and any Facebook format parser should be adapted easily to the artifact's format. In general, the data input is extensible.

In summary, it can be said that the contribution of this research is twofold. First, it provides an exploratory social network observatory. Essential information and challenges were discovered and a robust error handling introduced. Second, a comprehensive solution for retrieving a new market perspective from the customer point of view was presented focusing on Facebook data. Additionally, the information contained within should provide guidelines and a solid base for conducting further social network research and for creating further social observatories. With internet services and online social network services developing at a rapid pace and more and more services being created the possibilities of facilitating the data which they collect stays an interesting topic of research. It remains to be seen whether or not more services will open up their platforms and provide access to at least some part of their data warehouses giving academic researchers and in particular social scientists new ways of studying people's behavior and get a new perspective on markets.

2 References

Burnap, P. et al., 2014. COSMOS : Towards an integrated and scalable service for analysing social media on demand. *International Journal of Parallel, emergent and Distributed Computing*, pp.37–41.

Cioffi-Revilla, C., 2014. *Introduction to Computational Social Science*, Berlin: Springer Texts in Computer Science.

Hackenberg, R., 1970. The Social Observatory : Time series data for health and behavioral research. *Social Science and Medicine*, 4, pp.343–357.

Lasswell, H.D., 1967. Do We Need Social Observatories? *The Saturday Review*, pp.49–52.