Open-source Monitoring, Search and Analytics over Social Media

Manos Schinas[1,2], Symeon Papadopoulos[1], Lazaros Apostolidis[1], Yiannis Kompatsiaris[1], and Pericles A. Mitkas[2]

[1] Centre for Research and Technology Hellas, Thessaloniki, Greece
{manosetro,papadop,laaposto,ikom}@iti.gr
[2] Aristotle University of Thessaloniki, Greece
mitkas@auth.gr

Abstract. The paper describes a technical demonstration of an open-source framework for monitoring, analysis and search over multiple social media platforms. The framework is intended to be a valuable tool for media intelligence professionals, as well as a framework and testbed for scientists and developers with interest in social media research.

Keywords: social media, data collection, data mining, information retrieval, stream processing, information visualization

1 Introduction

Social media [1] is nowadays a key channel for communication, self-expression, information gathering and entertainment for billions of Internet users worldwide. The penetration of social media has consistently increased throughout the last decade and it has nowadays reached a point where the large majority of people worldwide regularly use one or more of numerous popular platforms for a wide variety of purposes. As a result, social media is often regarded as a sensor of real-world trends and events [2] and as an indispensable tool for performing social science research [3] and consumer intelligence gathering and marketing [4].

Existing social media research is typically based on one of two approaches depending on the setting where it is carried out. In *academic* settings, researchers typically use a variety of tools, scripts or libraries, and often develop their own additional custom scripts to perform the required data collection, manipulation and analysis. In *business* settings, analysts use a variety of software-as-a-service products, typically through a dashboard-like interface, offering access to statistics and analytics about queries of interest. The main disadvantage of the first approach is the increased effort that is necessary to set up the data collection and analysis pipeline. In contrast, the second approach suffers from limited flexibility, cost (most social media SaaS offerings are subscription-based) and dependence on proprietary solutions.

To address these limitations, we present an integrated open-source framework for monitoring, analyzing and retrieving social media content. The framework is easy to install locally and can then be managed as a set of services that expose data collection, indexing and retrieval capabilities via a REST API and a web-based user interface. In addition, the framework is designed in a modular way, which makes it straightforward to adapt and extend for different requirements, e.g. collect data from additional social media platforms, filter incoming content, compute additional metrics, etc. To our knowledge, the proposed framework is the only integrated open-source solution for social media monitoring, analytics and search, which is freely available and offers at the same time ease-of-use, rich off-the-shelf features, flexibility and extensibility.

The development of the presented framework started in SocialSensor[3], where a need came up to retrieve multimedia content around trending topics that were automatically detected by the news monitoring application developed within the project [5]. The development of the tool continued in REVEAL[4], while the current version presented in this paper is based on extensions made to support the social media monitoring needs of the STEP[5] and hackAIR[6] research projects.

2 Design and Implementation

The presented framework is built upon a set of projects, all of which are available on GitHub[7]. The core project (*mklab-framework-common*) contains a set of classes for defining a common data model across different social media platforms (Figure 1). Social media abstractions (*mklab-socialmedia-abstractions*) implements the data model for each of the supported social media platforms, e.g. Twitter, Facebook, etc., and contains a set of wrappers that encapsulate the different APIs provided by the platforms for the collection of data. The client project (*mklab-framework-client*) contains a set of classes for data management, including wrappers around different storage solutions, such as MongoDB and Apache Solr. Finally, the Stream Manager (*mklab-stream-manager*) incorporates all the orchestration and operational logic for data collection and storage.

2.1 Data model

Figure 1 depicts the data model used to store data in the platform. The left side of the Entity-Relationship (ER) diagram depicts objects collected by the platform: *items*, *media items*, *web pages* and *users*. Items correspond to the messages posted to the various social media platforms, e.g., status updates on Twitter, posts on Facebook, video posts on YouTube, etc. Media items correspond to the multimedia content that is embedded in these items, for instance, images in tweets, videos in Facebook posts, etc. Web pages correspond to the URLs that are contained in the items. Finally, users are the objects representing the user accounts publishing the items. The right side of the diagram depicts the *collection*

[3] `http://socialsensor.eu/`
[4] `https://revealproject.eu/`
[5] `http://step4youth.eu/`
[6] `http://www.hackair.eu/`
[7] `https://github.com/MKLab-ITI/`

entity, which offers users of the platform a way to organize social media content. A collection can comprise multiple queries as described in section 2.2, while the same query can be included in more than one collections. The platform supports three types of query: a) keyword queries, b) account queries, and c) location queries. It is important to note that there is no explicit (stored) association between a collection and the collected content (items, media items, etc.), nor is there such association between content and queries. This is depicted in the ER diagram by the dotted relationships between the corresponding entities.

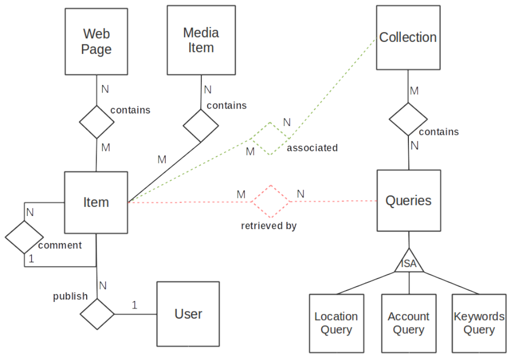

Fig. 1. Entity relationship data model of social media monitoring tool.

2.2 Data collection and processing

The collection of social media data is performed by the Stream Manager module. Currently, five different sources are supported: Twitter, Facebook, YouTube, Google+ and Flickr. The Stream Manager collects posts of interest (as defined by user queries) shared in these platforms alongside the users that published them, the embedded media items and the linked URLs. In addition to social media sources, the platform also supports monitoring RSS feeds.

Data collection by the Stream Manager is configured on the basis of user-defined collections, which, as mentioned in Section 2.1, consist of a set of three types of query: keywords, accounts, locations. Keywords can be defined on the basis of simple keywords, e.g. *"air pollution"*, or more complex logical expressions, e.g. *"election results" AND ("donald trump" OR "hilary clinton")*. Accounts refer to public sources of content for each platform, e.g., Twitter users, Facebook pages, YouTube channels, etc. Locations are represented as bounding boxes defined by pairs of latitude/longitude coordinates.

Given a collection, the associated queries are translated to the appropriate API calls. For example if a Twitter account such as *@BBC* is specified as one of the queries in the collection, that account is mapped to a call to the Twitter API that is used for the collection of Tweets posted by the specific user. In the same way, given a logical expression of keywords, multiple API calls are generated, one for each of the supported sources. However, given that the tool may be used by several end users, several of the defined collections may contain identical queries, e.g. the same keywords. To make the query process more efficient, the Stream Manager first de-duplicates all input queries into unique non-redundant query elements. Then, the Stream Manager periodically polls each of these unique query elements, and keeps track of the number of submitted requests to each platform in order to respect the limits imposed by it. It is worth noting that the set of Stream Manager queries may change dynamically (e.g., when a user creates or updates a collection), and such changes are efficiently communicated to the Stream Manager through a Redis[8] message broker instance.

The fetched items are then processed by a sequence of *filters* and *processors*, executed within the Stream Manager, before being stored and indexed. In terms of filtering, a set of heuristic rules is applied to keep only items of high quality. For example, items with limited text content, or with too many hashtags and URLs are treated as spam messages and therefore discarded. In terms of processing modules, three indicative processors are provided off-the-shelf by the Stream Manager: language detection, named entity extraction, and MinHash signature extraction from the text of the item. Finally, the collected content elements (items, media items, users and web pages) are stored in a MongoDB[9] instance.

2.3 Indexing and retrieval

Data indexing is based on Apache Solr[10]. For each item, a subset of its fields are indexed. This includes all the textual fields, e.g. title, and other relevant fields, e.g., publication time, user id, number of views, etc., which can be used for filtering and faceting. To retrieve content related to a specific collection a Solr query is generated based on the queries that are associated with the collection, and the ids of the relevant items are retrieved. Then, the corresponding item metadata and all associated entities are retrieved from MongoDB by id.

The platform analytics capabilities range from simple metrics such as the number of items or unique users, to more complex ones such as *reach* (estimated cumulative audience for a set of items) and *endorsement* (estimated sum of "likes" for a set of items). Top users, locations, tags and named entities (based on frequency of appearance) are also provided. Finally, the platform supports the generation and visualization of timelines with different time granularities.

Analytics operations are implemented on top of two Solr components: Faceting and Stats. For example, for the generation of the top active users in the collection (in terms of number of posted items), we make a facet request in Solr, using the user id as the field to be treated as a facet. The corresponding response by Solr contains the top N users along with the number of items per user. In a similar manner, using the Stats component we can calculate field-oriented statistics, e.g. the average number of views or sum of shares per collection.

[8] `https://redis.io/`
[9] `https://www.mongodb.com/`
[10] `http://lucene.apache.org/solr/`

Finally, the analytics framework leverages two more Solr components to make easier the exploration of collections: Result Clustering and Result Collapse. Clustering is used to automatically discover groups of similar search hits, i.e. groups of items related to a specific topic. These identified topics can be used to narrow down the set of items associated with a collection. Collapse, on the other hand, is used to group search results based on the value of a field. In our case, we use collapse based on the MinHash signature of items to support de-duplication of content, i.e. items with the same signature collapse into a single item.

Figure 2 depicts the key software components of the framework along with the function that each performs. Further details regarding their deployment are provided in section 3.3.

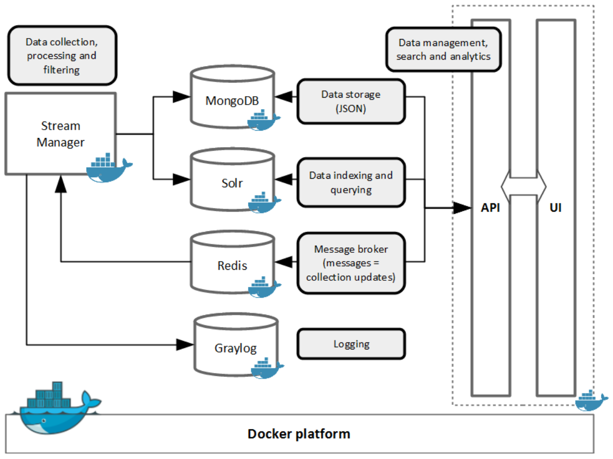
Data collection, processing and filtering
Stream Manager
MongoDB
Solr
Redis
Graylog
Data storage (JSON)
Data indexing and querying
Message broker (messages = collection updates)
Logging
Data management, search and analytics
API
UI
Docker platform

Fig. 2. Overview of framework components (white blocks) and their function (gray).

3 Usage and Deployment

3.1 Application Programming Interface

All the data collection, indexing, aggregation and retrieval capabilities of the framework are exposed to developers through a REST API that is built on PHP and served by an Apache web server. The methods of the API belong to three categories: a) collection management, b) content retrieval and c) analytics. The first category contains a set of methods to create/update/delete collections and to retrieve collections created by a specific end user. For content retrieval there is the */items/collection-id* method, which is used to get the items related to a specific collection. The method has a set of parameters to refine how retrieval takes place, e.g. filtering or sorting. Finally, analytics over a selected collection is exposed through six methods: */statistics*, */timeline*, */users*, */terms*, */countries* and */heatmap/points*. For each of these methods the unique identifier of the collection must be specified. In addition, a set of parameters for the refinement of the collection (e.g. by source or language) is also provided.

3.2 User interface

The framework user interface consists of three main pages: a) a page where users of the platform can create their own collections, b) a feed view for browsing content (Figure 3), and c) a dashboard view for analytics (Figure 4).

The feed view presents the list of media items collected in the form of a stream. Each item in the view features relevant metadata, namely the publication time, username and social media platform. The feed can be presented in both gallery/grid (Figure 3) and list form. The user can also search for items based on free text queries and use a number of filtering and sorting criteria. The following filters can be applied: a) source (social media platform), b) language, c) topic, d) original (show retweets/shares or not), e) type (text item or item with embedded multimedia), f) unique (remove near-duplicate content), g) date. In addition to filtering, end users can rate the items retrieved for a collection on a scale of 1-5 (1: "irrelevant", 5: "relevant"). In addition, users may choose to exclude selected items and/or users along with their published items.

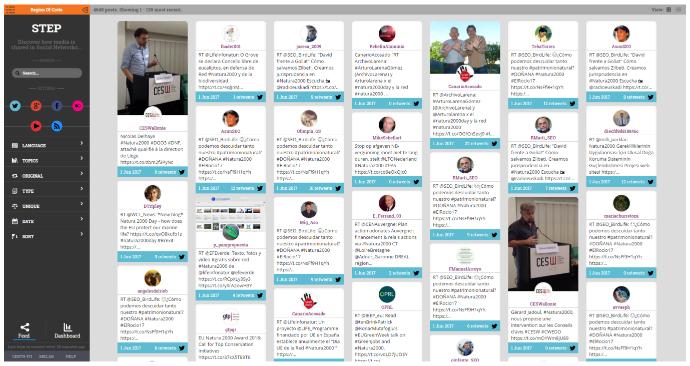

Fig. 3. Snapshot of feed gallery view.

The dashboard view offers metrics and widgets that depict summary views over the collected data. The visualizations are dynamic by leveraging the same set of filters as the ones in the feed view. In more detail, the dashboard consists of the following widgets:

– Numbers of posts made, users talking, users reached and endorsements, and platform contribution pie chart.
– Heatmap based on the exact location (latitude, longitude) of geo-tagged items in the collection (typically a small subset of all items).
– User location at the level of a country, which is automatically detected from the location text field in users' profiles.
– Timeline visualization based on the number of items over time. The granularity of the depiction may be set to hour, day or week.
– Top N users with most posts in the collection. N can be set between 10 and 200. Avatar, username and total number of posts for each user is presented along with the link to their social profile pages.

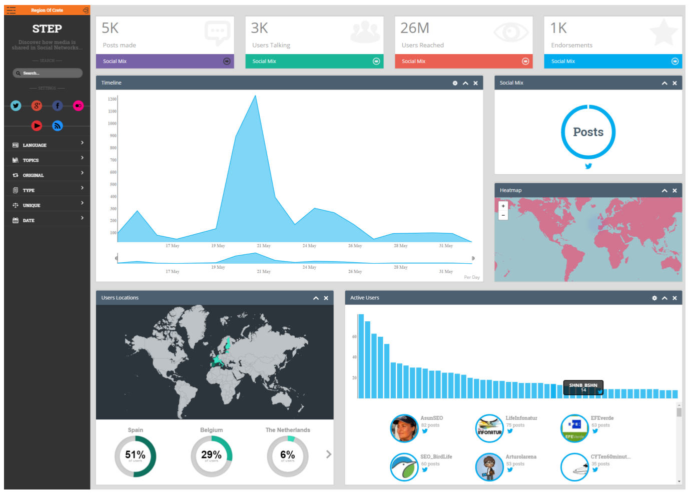

Fig. 4. Snapshot of Dashboard view.

– Top N entities in terms of frequency. N can be set between 10 and 200. Entities are organized in three frequency categories (often, occasionally, seldom) and three types (person, tag, location).

3.3 Deployment to Local Environment

For deploying the framework as a locally administered service, we opted for the use of Docker[11]. Each of the framework components is deployed as a separate Docker container[12]. Figure 2 depicts all components and the communication between them. For MongoDB and Solr, the official Docker images are used. For the Stream Manager, we build an image that clones a stable release of the module from its GitHub repository and generates an executable jar using maven. The REST API and user interface is built and deployed as a separate Docker image on top of Apache web server. Finally, the signaling of collection-related events to MongoDB is handled by a Redis instance, while the aggregation and storage of logs by a Graylog instance[13], both deployed on top of Docker.

Using Docker Compose[14], we configure all components in a single YAML[15] file. This only needs limited edits to be adapted to users' local environment. The only requirement is to set the directory paths for MongoDB, Solr, etc. to the local file system. Optionally, service ports can be edited to values other than the default ones. In addition to the YAML configuration file, a separate configuration file associated with the Stream Manager needs to be edited to enable the use of social media APIs, by providing user credentials for each platform.

[11] `https://www.docker.com`
[12] Available on: `https://github.com/MKLab-ITI/mmdemo-dockerized`
[13] `https://www.graylog.org/`
[14] `https://docs.docker.com/compose/`
[15] `http://yaml.org/`

4 Limitations and Future Work

There are three main limitations in the usage of the presented framework. A first limitation is due to API usage restrictions imposed by the social media platforms. For example, Twitter restricts the number of allowed calls in a time window of 15 minutes[16]. Other platforms have similar restrictions but with different time granularities. As the number of collections and associated queries increases, it is necessary to decrease the request rate for each query. This creates a delay in the discovery of new content, especially for queries (keywords) that exhibit high activity. Moreover, a limitation pertains to the large scale of the collected content items, which may easily reach the order of tens of millions. As the association of content to collections is dynamically resolved based on the Solr indexing mechanism, the response time of the tool is adversely affected by the size of Solr index. This is more pronounced in the case of analytics, which are calculated on the fly using the faceting mechanism of Solr. A third limitation is the lack of built-in support in the framework for managing the privacy risks that are inherent when someone performs data collection from social media. In particular, the framework does not support secure data storage and access (e.g. via encryption), nor does it offer any facilities for removing user-generated content that could accidentally reveal personal information.

To improve the scalability of the framework, we plan to investigate and develop means of distributing the query and indexing load over multiple nodes (e.g., via sharding and replicating the index). In terms of functionality, we aim at two further improvements: a) more sophisticated relevance models that go beyond the simple heuristic rules that are currently used for filtering, b) support for additional social media platforms such as Sina Weibo and VKontakte.

Acknowledgements. This work was supported by the STEP (H2020-649493) and HackAir (H2020-688363) projects, funded by the European Commission.

References

1. Kaplan, A. M., Haenlein, M.: Users of the World, Unite! The Challenges and Opportunities of Social Media. Business Horizons 53(1), 59-68 (2010)
2. Aiello, L.M., Petkos, G., Martin, C., Corney, D., Papadopoulos, S., Skraba, R., Goker, A., Kompatsiaris, Y., Jaimes, A.: Sensing Trending Topics in Twitter. IEEE Transactions on Multimedia 15(6), 1268-1282 (2013)
3. Mejova, Y., Weber, I., Macy, M.W.: Twitter: a Digital Socioscope. Cambridge University Press (2015)
4. Fan, W., Gordon, M.D.: The Power of Social Media Analytics. Communications of the ACM 57(6), 74-81 (2014)
5. Diplaris, S., Papadopoulos, S., Kompatsiaris, I., Goker, A., Macfarlane, A., Spangenberg, J., Hacid, H., Maknavicius, L., Klusch, M.: Socialsensor: sensing user generated input for improved media discovery and experience. Proceedings of the 21st Int. Conference on World Wide Web (WWW '12 Companion), 243-246, ACM (2012)

[16] `https://dev.twitter.com/rest/public/rate-limiting`