FP7-ICT Strategic Targeted Research Project TrendMiner (No. 287863)

Large-scale, Cross-lingual Trend Mining and Summarisation of Real-time Media Streams

D5.1.1 Real-time Stream Media Collection - V1

Philippe Rigaux (IMR)

Abstract

FP7-ICT Strategic Targeted Research Project TrendMiner (No. 287863)

Deliverable D5.1.1 (WP 5)

This deliverable describes the infrastructure and analytic framework set up by Internet Memory during the initial phase of the TrendMiner project. Section 1 and 2 are respectively devoted to the infrastructure (including the software components) and the set of tools that lets us crawl, ingest, classify and extract data relevant to the project. Based on this setting, we initialized the constitution of a collection oriented towards the TrendMiner objectives and covering some representative excerpt of social media content. We briefly present this initial harvesting and extraction work in Section 3.

This deliverable is complementary to the D5.2 TrendMiner deliverable devoted to the TrendMiner architecture (see in particular the chapter covering HBase and HDFS).

Keyword list: real-time stream media collection

Nature: Other Dissemination: PU
Contractual date of delivery: M12 Actual date of delivery: M12
Reviewed By: DFKI, SOTON
Web links: Mignify API (http://im1c7.internetmemory.org:8080/)

CHANGES

Version	Date	Author	Changes
1.0	22.10.2012	Philippe Rigaux	First version
	25.10	Thierry Declerck	Internal review, adding comments
1.1	05.11	Philippe Rigaux	Revision following Thierry's comments

TrendMiner Consortium

This document is part of the TrendMiner research project (No. 287863), partially funded by the FP7-ICT Programme.

DFKI GmbH
Language Technology Lab
Stuhlsatzenhausweg 3
D-66123 Saarbrcken
Germany
Contact person: Thierry Declerck
E-mail: declerck@dfki.de

University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP
UK
Tel: +44 114 222 1930
Fax: +44 114 222 1810
Contact person: Kalina Bontcheva
E-mail: K.Bontcheva@dcs.shef.ac.uk

University of Southampton
Southampton SO17 1BJ
UK
Contact person: Mahensan Niranjan
E-mail: mn@ecs.soton.ac.uk

Ontotext AD
Polygraphia Office Center fl.4,
47A Tsarigradsko Shosse,
Sofia 1504, Bulgaria
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

Internet Memory Research
45 ter rue de la Rvolution
F-93100 Montreuil
France
Contact person: France Lafarges
E-mail: contact@internetmemory.org

Sora Ogris and Hofinger GmbH
Bennogassee 8/2/16
1080 Wien
Austria
Contact person: Christoph Hofinger
E-mail: ch@sora.at

Eurokleis S.R.L.
Via Giorgio Baglivi, 3
Roma RM
00161 Italy
Contact person: Francesco Bellini
E-mail: info@eurokleis.com

Hardik Fintrade Pvt Ltd.
227, Shree Ram Cloth Market,
Opposite Manilal Mansion,
Revdi Bazar, Ahmedabad 380002
India
Contact person: Suresh Aswani
E-mail:m.aswani@hardikgroup.com

Executive Summary

This deliverable describes the infrastructure and analytic framework set up by Internet Memory during the initial phase of the TrendMiner project. Section 1 and 2 are respectively devoted to the infrastructure (including the software components) and the set of tools that lets us crawl, ingest, classify and extract data relevant to the project. Based on this setting, we initialized the constitution of a collection oriented towards the TrendMiner objectives and covering some representative excerpt of social media content. We briefly present this initial harvesting and extraction work in Section 3.

This deliverable is complementary to the D5.2 TrendMiner deliverable devoted to the TrendMiner architecture (see in particular the chapter covering HBase and HDFS).

List of Acronyms

API	Application Programming Interface
ETL	Extract, Transform Load
HDFS	Hadoop Distributed File System
HBase	HBase, an Open source implementation of Google's BigTable
MIME	Multipurpose Internet Mail Extensions
NLP	Natural Language Processing
ODM	Object-Document Model
REST	REpresentational State Transfer
RSS	Rich Site Summary

Contents

Table of figures

1 Description of the architecture

Our infrastructure relies on a data center hosted by Internet Memory. It contains a continuously expanding set of servers designed for reliable long-term storage of big datasets. The software components chosen to manage these datasets is the well-known Hadoop suite, with two prominent layers:

1. HDFS (Hadoop Distributed File System, http://hadoop.apache.org), a distributed file system natively equipped with replication, load balancing and fault tolerance features;
2. HBase (http://hbase.apache.org), a persistent distributed data structure with indexing capabilities, designed for very large datasets (TeraBytes or even PetaBytes). At the HBase level, collections of Web resources are organized as tables.

On top of this architecture, Internet Memory provides a platform called **Mignify**, supporting the execution of analytic workflows on Big Data. The following figure (Figure 1) shows the main components of the platform.

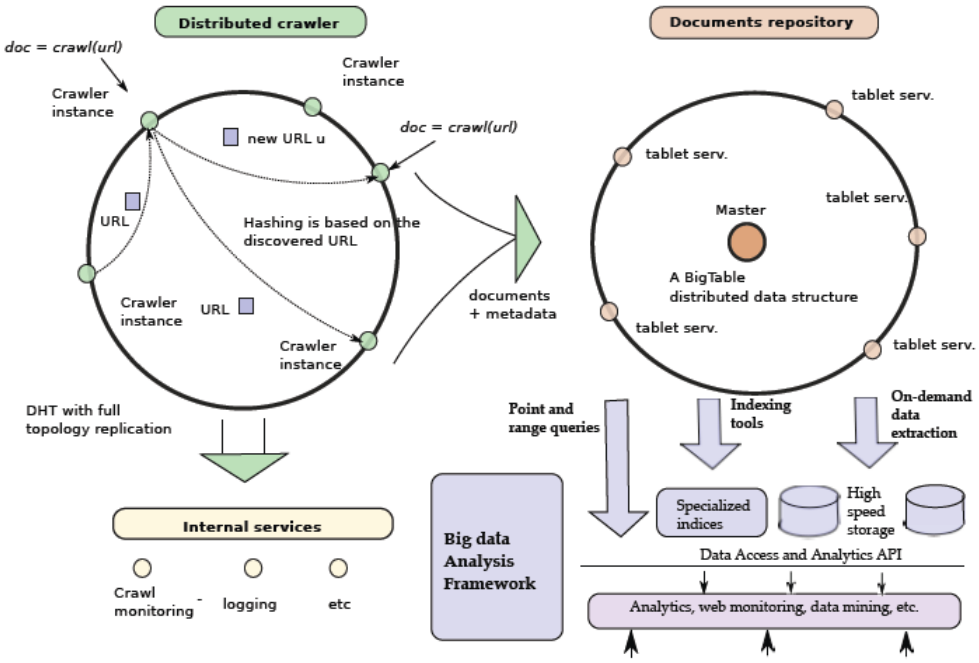


Figure 1: Mignify platform schema

1. the left part is a crawler, MemoryBot, designed to operate at large scale;
2. the top-right part is the Hadoop/HBase repository where datasets are persistently stored;
3. the bottom-right part is the Mignify extraction platform.

The crawler is an internal tool and its design and operational features is not detailed here. It is important however to know that (i) MemoryBot is a distributed software, and it performance (number of resources collected per time unit) is proportional to the

number of machines (the “cluster”) assigned to a crawl; (ii) MemoryBot does not directly insert in the repository, but rather produces compressed archive files which are subsequently “ingested” in Hadoop/HBase.

The ingestion process is a distributed process composed of a set of tasks operating in parallel. Each task takes a compressed archive file, extracts Web resources, and inserts these resources in a HBase table. During an ingestion, analytic operations can be applied either to a single resource or to group of resources. These operations are specified in Mignify with a rich of abstractions (filters, extractors, views) which are detailed in the next section.

The representation of a *resource* (or document) in Mignify is complex. First, they are identified by their URL (the *rowkey* in HBase). Second, we store and preserve several *versions* of a resource, each tagged by a timestamp. MemoryBot indeed repeatedly scans the Web, searching for updates of resource. The temporal stack of versions is a distinctive feature of Mignify. Finally, we store not only the mere content of a resource, but also associated features which are either obtained from the Web server during the crawl (e.g., MIME type, length, HTTP fields) or *extracted* during the ingestion. A standard feature is for instance the MIME type of the document. More sophisticated extractors may produce for instance, for a textual document, features like entity references or sentiment annotations.

In the context of TrendMiner we are mostly interested in so-called *active* sources, i.e., Web content which evolves quite often and reflect the activity of a community around a particular area of interest (e.g., political, financial, in the context of the 2 use cases of TrendMiner: “Multilingual Trend Mining and Summarisation for Financial Decision Support” and “Multilingual public spheres: political trends and summaries “). Active source types cover, but are not limited to, RSS feeds, blogs, forums, Twitter accounts, etc.

In order to properly represent the complex object which constituted a resource, Mignify encapsulates its representation with an *Object-Document Model (ODM)*. The Java API maps the HBase row storage to the ODM which presents a consistent and simplified representation of resources. We provide to our Mignify partners the Java API, and plan to develop other means of accessing and querying the datasets stored in Mignify.

2 Processing ETL operations with Mignify

Let us now develop the support for analytics at large provided by Mignify. First, Mignify maintains a “Catalog” of *filters* and *extractors*. A *filter* determines the subset of a collection to which an *extractor* applies. For instance a filter can select RSS resources from the TrendMiner collection in order to apply some RSS-related extrator. An *extractor* takes a resource as input, and produces *features* which are associated to the resource (we call this the *annotation* process). The result of applying an extractor is therefore an annotated resource which can in turn be submitted to the filter-extractor mechanism. The process is illustrated below (Figure 2) .

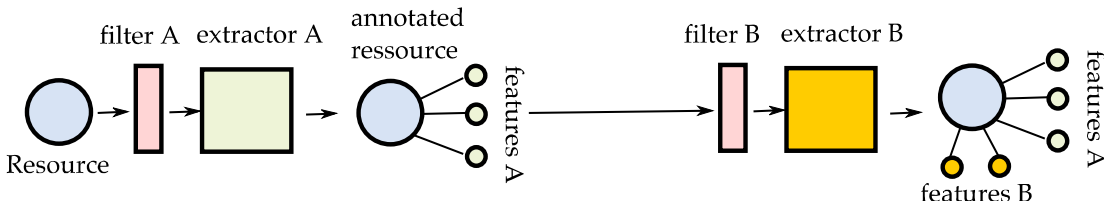
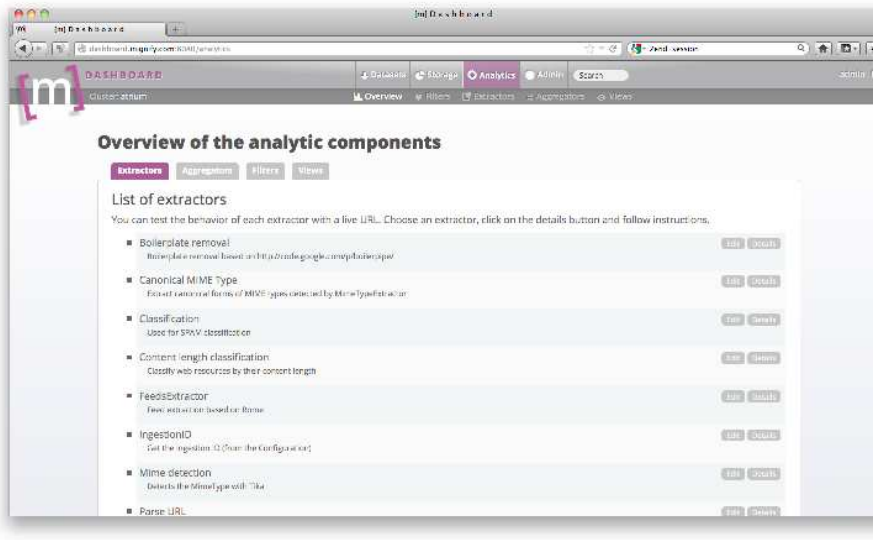


Figure 2: Annotation process

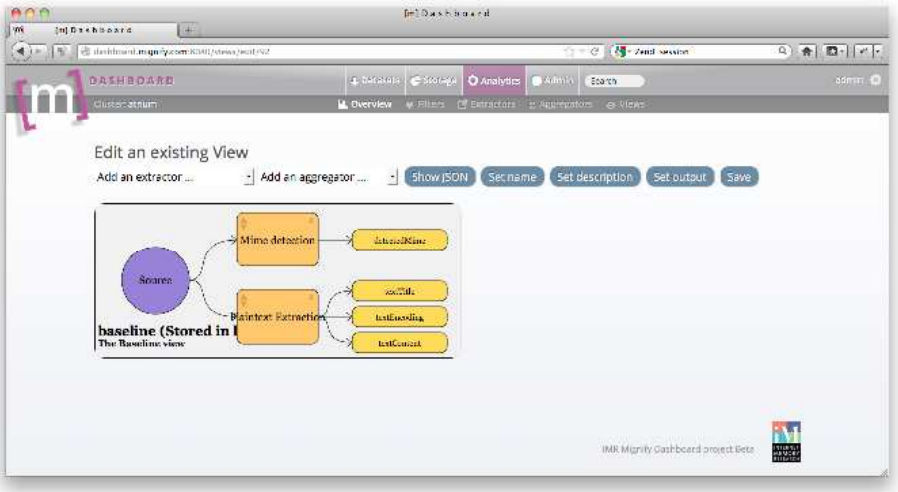
An extractor is a software component which implements a particular interface or API. It follows that Mignify's Catalog is easily extended with new extractors as long as they comply to the plaform API. TrendMiner partners specialized in analytic methods (say, sentiment analysis or other NLP operators) can therefore "plug" their extractors in Mignify in order to operate the extraction at scale in our infrastructure. The approach brings two major benefits

1. First, analytic components are brought near to the data, following the data locality principle (simply speaking, moving TBs of data is not an option);
2. Second, our partners benefit from Internet Memory expertise in applying data analytics to very large datasets.

The following screenshot shows the Web interface of Mignify, showing the extractor's Catalog.



Extractors can be applied to a Collection either when resources are inserted (ingestion) or as a post-processing job. Actually, we combine in Mignify filters and extractors in pipeline specifications called *views*. Views are the central mechanism through which data analysis is applied to datasets in Mignify. A view may refer to any extractor registered in the platform, and the Mignify interface lets users associate filters and extractors from the catalog to create views. The next screen shot shows the Mignify views editor. The specific example illustrated here is the so-called *baseline*, a views which applies some low-level extraction (MIME type, plain text) and is used with all collections.



In Mignify, we associate to each HBase table one or several *views*. As said above, the *baseline* view is always applied. For TrendMiner collections, we also apply the *RSS feeds* view which selects all RSS feeds from a general Web data collection, and extract the main features of these feeds (author, date, copyright, etc.). The ingestion process inspects these views when it starts, and applies the extractors to resources whenever necessary.

The following figure summarizes the association of a collection with views, and how views are processed during an ingestion job.

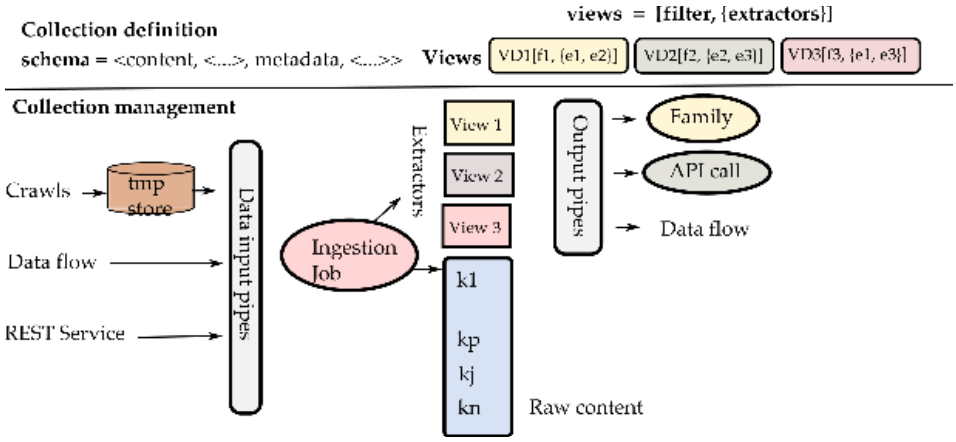


Figure 3: Collection with views

At the HBase level, collections of Web resources are organized as tables with two mandatory column families, `content` and `meta`, and optional column families containing extracted information, defined on a per-collection basis. In practice, a view specification attached to a collection most often gives rise to a dedicated column family in the HBase table storing the collection.

Let us focus on view specifications, expressed as a combination of filters and extractors. Turning back to the Figure 3, views are defined independently from collections but they can, once defined, be associated to a collection thanks to a

subscription mechanism. The lower level of the figure shows how the result of views is computed: an *extraction engine* (actually a generic MapReduce job[1]) observes any input affecting the content of a collection. A typical example is the production of Arc of Warc files[2] by a crawler, and the request to insert the content of these files in a given collection. The extraction engine then looks for the subscriptions attached to the collection, and computes the view content by applying the extraction workflow illustrated above to all input resources.

We designed the platform as an open space where new extractors can be added, as long as they comply to an interface defined in our core library. As said above, this means that TrendMiner partners can easily integrate their own components. The framework takes care of technical aspects related to the distribution of extractor execution in the framework.

3 Content Acquisition : RSS

We focused during the initial phase of the project on harvesting RSS feeds.

3.1 Crawling RSS feeds with MemoryBot

In order to cope with the requirements of social media acquisition, we extended our crawler to periodically recrawl a given list of RSS feeds. For TrendMiner, we use a dedicated crawler cluster which is able, at run time, to dynamically load a list of RSS feeds that need to be harvested. Recall that our ingestion engine processes a RSS feeds view which detects feeds from large web datasets. The view result is therefore a list of newly discovered feeds which can be sent back to the crawler.

3.2 Data set

We operated two crawls, both focused on RSS feeds harvesting, but based on different stategies:

- In the first one we run a crawl from a list of RSS feeds sent by the TrendMiner partner SORA (http://www.sora.at). The scope specifies a 1-hop link (i.e., we follow one link from feeds and collect the referred page). We do not apply discovery, and focus on the periodic refreshment of the sources.
- In the second one we run a global crawl on the Web of a large European Country (Italy). We found 1.2 Billion Web resources and filtered the RSS feeds from this big dataset. We found 100,000 such feeds. Note that in the moment we cannot assess the topics of these feeds : tagging feeds, or more generally active sources, with topic categorization is part of ongoing work.

We created a HBase table dedicated to the storage and access to the feeds and their extracted features. Mignify provides a Web interface to browse a table and inspect the statistics, ongoing ingestions, etc. The following screenshot (Figure 4) shows for instance a page giving general information about the TrendMiner table.

[1] See here the TrendMiner Deliverable D5.1.2: Architecture for Distributed Text Annotation and Trend Mining Over Media Streams (v.1) for more details.

[2] Both Arc and Warc are normalized file format for representing set of Web resources and store them as an archive. See http://archive-access.sourceforge.net/warc/.

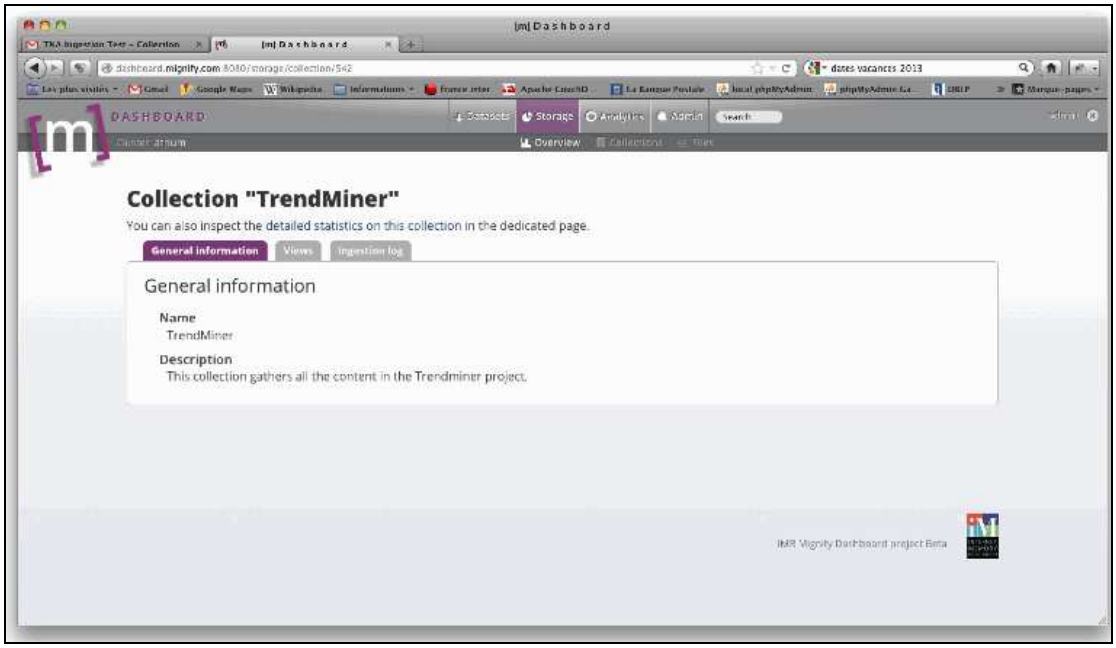


Figure 4: Mignify - Web interface

4 Next steps

This initial work allowed us to set up an environment adapted to the TrendMiner objective. We created a TrendMiner storage space in our Mignify platform, operated initial craws focused on RSS feeds, and started to ingest data of interest to the project.

Ongoing work aims at addressing three important issues: (i) classification of sources, (ii) extension of source types beyond RSS, and (iii) implementation of an easy access to the datasets for our partners.

The first issue will initially consider only feeds, and will aim at automatically detecting the feeds of interest to TrendMiner (either in the political or financial area). We are currently working on a general classifier for Web resources, and will apply this tool to the specific problem of recognizing the topic(s) of RSS feeds.

The second step ambition is to collect as “active” sources not only RSS feeds, but also blogs, forums, and generally any discussion area where information of interest can be found.

Finally, the Mignify datasets are currently accessible via a Java API. We want to go further and develop a more sophisticated interface, allowing queries, filtering and restructuration of results, as well as access through a REST communication.