Project Acronym	***SoBigData***
Project Title	***SoBigData Research Infrastructure*** ***Social Mining & Big Data Ecosystem***
Project Number	***654024***
Deliverable Title	***Resource adaptation to register to the e-infrastructure 3***
Deliverable No.	***D10.10***
Delivery Date	***January 2019***
Authors	***Massimiliano Assante (CNR), Leonardo Candela (CNR), Paolo Manghi (CNR), Pasquale Pagano (CNR)***

SoBigData receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654024

DOCUMENT INFORMATION

PROJECT	
Project Acronym	SoBigData
Project Title	SoBigData Research Infrastructure Social Mining & Big Data Ecosystem
Project Start	1st September 2015
Project Duration	48 months
Funding	H2020-INFRAIA-2014-2015
Grant Agreement No.	654024
	DOCUMENT
Deliverable No.	D10.10
Deliverable Title	Resource adaptation to register to the e-infrastructure 3
Contractual Delivery Date	30 November 2018
Actual Delivery Date	23 January 2019
Author(s)	Massimiliano Assante (CNR), Leonardo Candela(CNR), Paolo Manghi, (CNR) Pasquale Pagano (CNR)
Editor(s)	Massimiliano Assante (CNR), Beatrice Rapisarda (CNR)
Reviewer(s)	Roberto Trasarti (CNR), Beatrice Rapisarda (CNR)
Contributor(s)	Kalina Bontcheva (USFD), Marco Cornolti (UNIPI), Stefano Cresci (CNR), Thorsten May (FRH), Cristina Muntean (CNR), Marco Ponza (UNIPI), Daniele Regoli (SNS), Roberto Trasarti (CNR)
Work Package No.	WP10
Work Package Title	JRA3_SoBigData e-Infrastructure
Work Package Leader	CNR
Work Package Participants	CNR, USFD, UNIPI, FRH, UT, IMT, LUH, KCL, SNS, AALTO, ETHZ
Dissemination	Public
Nature	Report
Version / Revision	1.0
Draft / Final	final
Total No. Pages (including cover)	26
Keywords	e-infrastructure, services, resources, integration

DISCLAIMER

SoBigData (654024) is a Research and Innovation Action (RIA) funded by the European Commission under the Horizon 2020 research and innovation programme.

SoBigData proposes to create the Social Mining & Big Data Ecosystem: a research infrastructure (RI) providing an integrated ecosystem for ethic-sensitive scientific discoveries and advanced applications of social data mining on the various dimensions of social life, as recorded by “big data”. Building on several established national infrastructures, SoBigData will open up new research avenues in multiple research fields, including mathematics, ICT, and human, social and economic sciences, by enabling easy comparison, re-use and integration of state-of-the-art big social data, methods, and services, into new research.

This document contains information on SoBigData core activities, findings and outcomes and it may also contain contributions from distinguished experts who contribute as SoBigData Board members. Any reference to content in this document should clearly indicate the authors, source, organisation and publication date.

The document has been produced with the funding of the European Commission. The content of this publication is the sole responsibility of the SoBigData Consortium and its experts, and it cannot be considered to reflect the views of the European Commission. The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated the creation and publication of this document hold any sort of responsibility that might occur as a result of using its content.

The European Union (EU) was established in accordance with the Treaty on the European Union (Maastricht). There are currently 27 member states of the European Union. It is based on the European Communities and the member states’ cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice, and the Court of Auditors (http://europa.eu.int/).

GLOSSARY

ABBREVIATION	DEFINITION
VA	Virtual Access
VRE	Virtual Research Environment
WPS	Web Processing Service

TABLE OF CONTENT

DELIVERABLE SUMMARY

Deliverable D10.10 “Resource adaptation to register to the e-infrastructure 3” is the revised version of the Deliverable D10.9 “Resource adaptation to register to the e-infrastructure 2” and Deliverable D10.8 “Resource adaptation to register to the e-infrastructure 1” intended to report the experiences of partners from different infrastructures at integrating their services, methods, and applications as SoBigData resources. The first section describes the general integration patterns, while the second section reports the experiences from the individual partners, revealing the *effort required*, in terms of time and technical complexity, and *earned benefits*. This revised version of the document covers the whole period of the project, including the up to date information of the D10.9 deliverable and the new experiences of partners at integrating their services, methods, and applications as SoBigData resources, developed through the project’s lifetime.

EXECUTIVE SUMMARY

Deliverable D10.10 “Resource adaptation to register to the e-infrastructure 3” is the revised version of the Deliverable D10.9 “Resource adaptation to register to the e-infrastructure 2” and Deliverable D10.8 “Resource adaptation to register to the e-infrastructure 1”. In order to achieve Virtual Access SoBigData partners must undertake a process of integration of their method resources in the e-infrastructure. This deliverable reports the experiences of partners from different infrastructures at integrating their services, methods, and applications as SoBigData resources. The first section describes the general integration patterns, while the second section reports the experiences from the individual partners, revealing the *effort required*, in terms of time and technical complexity, and *earned benefits*.

The feedback obtained during the first and second reporting period has been used to further simplify the process of integration and mitigate complexity (when this can be done). Specifically, to facilitate the importing process of methods (and their version updating) in the SoBigData infrastructure, as already reported in D10.9, a new Web application named SAI (Statistical Algorithms Importer) has been made available. SAI acts as a Method Importer for the SoBigData infrastructure.

At the time of its release, the deliverable covers the activities in the addressed areas performed in the first 41 months of the project. The project activities cover all effort spent from partners at integrating their services, methods, and applications as SoBigData resources, developed through the project’s lifetime. The deliverable was intially planned at M39, after 2 months from its previous version (D10.9). Being such short time from one version to the other we decided to deliver D10.10 at M41 (January 2019). We used the two additional months to complete the integration and report on (i) the WAT (cf. 4.1.4) and SWAT (cf. 4.1.5) applications, which were only partially integrated at M39 and, (ii) on the additional 34 GATE Cloud methods (cf. 4.2.3) as well as the SCube method (cf. 4.2.6).

1 RELEVANCE TO SOBIGDATA

In order to achieve Virtual Access SoBigData partners must undertake a process of integration of their method resources in the e-infrastructure. This document reports on the actual experience of scientists involved in this process, with the aim of highlighting the challenges and the benefits. This feedback will be used to further simplify the process of integration and mitigate complexity when this can be done.

2 INTRODUCTION

This deliverable reports the experiences of partners from different infrastructures at integrating their services, methods, and applications as SoBigData resources. It is the revised version of the document covering the whole period of the project, including the up to date information of the D10.8 deliverable and the new experiences of partners at integrating their services, methods, and applications as SoBigData resources, developed through the project's lifetime.

The first section describes the general integration patterns, which were improved with respect to D10.8. In particular, to facilitate the importing process of methods (and their version updating) in the SoBigData infrastructure, a new Web application named SAI (Statistical Algorithms Importer) has been made available. SAI provides scientists with an interface that allows to easily and quickly import Java, Python or R scripts onto the general-purpose gCube-based data analytics engine.

The second section reports the experiences from the individual partners, revealing the *effort required*, in terms of time and technical complexity, and *earned benefits*.

3 INTEGRATION PATTERNS

3.1 APPLICATIONS

An application in the SoBigData infrastructure is a stand-alone system running on a remote server and offering one or more social mining methods via WebUIs; in some cases it may also offer social mining datasets.

A lightweight integration is achieved by explicitly registering/publishing application information on the SoBigData catalogue. Each application is described in the catalogue by a record enabling its discovery and including a URL to the web application.

A mild integration consists in integrating the WebUIs as a portlet in the SoBigData VREs, leaving the application running on the remote server, equipped with SmartGears (see Wiki page) to account access to methods from SoBigData users. To achieve a deeper integration, the action can include the integration of the application with the VRE workspace, allowing scientists to provide input to the application directly from the local workspace and expect the results of the application to be stored in the local workspace.

A full integration consists instead in:

- Reconsidering application's architecture in order to extrapolate the methods and the datasets;
- Make methods compliant with the guidelines of the hosting platform according to the methodology described in a dedicated Wiki page. This activity might require some modification / adaptation of the method implementation, e.g. for input parameters specification. The cost of this adaptation depends on the complexity of the method;
- A SoBigData: step-by-step procedure for algorithm integration (including a couple of Java simple classes) resulting from a concrete integration exercise is available;
- Publish the algorithm through the platform. In case the method is implemented with a R script, the platform is provided with a facility supporting this publishing phase;
- Transform the datasets in publishable assets and publish them as previously described;

Once "fully integrated", the application actually becomes a set of SoBigData social mining assets that will benefit from:

- **Scalability** Will benefit from a distributed and scalable computing platform;
- **Repurposing** Can be exploited in the context of many virtual research environments and it is suitable for being repurposed / applied to datasets;
- **Standard accessibility** Will be automatically made available via a web-based GUI as well as with web-based protocols (SOAP and Rest);
- **Accounting** Are monitored and assessed by SoBigData tools, e.g. detailed statistics on usage are transparently collected.

If "mildly integrated" the application will only benefit from **repurposing** and **accounting** benefits.

3.2 METHODS

A method in the SoBigData infrastructure is a piece of code in Java, Python or R that implements a social mining algorithm / procedure.

Besides a **lightweight integration,** that can be achieved by explicitly registering/publishing method software information via the SoBigData infrastructure resource catalogue (where each method software is described in the catalogue by a record enabling its discovery and usage and including a URL to the software), during the last period effort has been spent to facilitate the importing process of new methods in the SoBigData infrastructure.

As a result of this effort a new Web application named SAI (Statistical Algorithms Importer) has been made available in the SoBigData infrastructure. SAI complements the **full integration** pattern (achieved by "wrapping" (implement the relative APIs) the method as a WPS method) by providing scientists with an interface that allows to easily and quickly import Java, Python or R scripts onto the general-purpose [gCube-based data analytics]() engine. Additionally, it allows scientists to update their scripts without following long software re-deploying procedures previously necessary.

The SAI Web application interface (Figure 1) resembles the R Studio environment, a popular IDE for R scripts, in order to make it friendly to script providers.

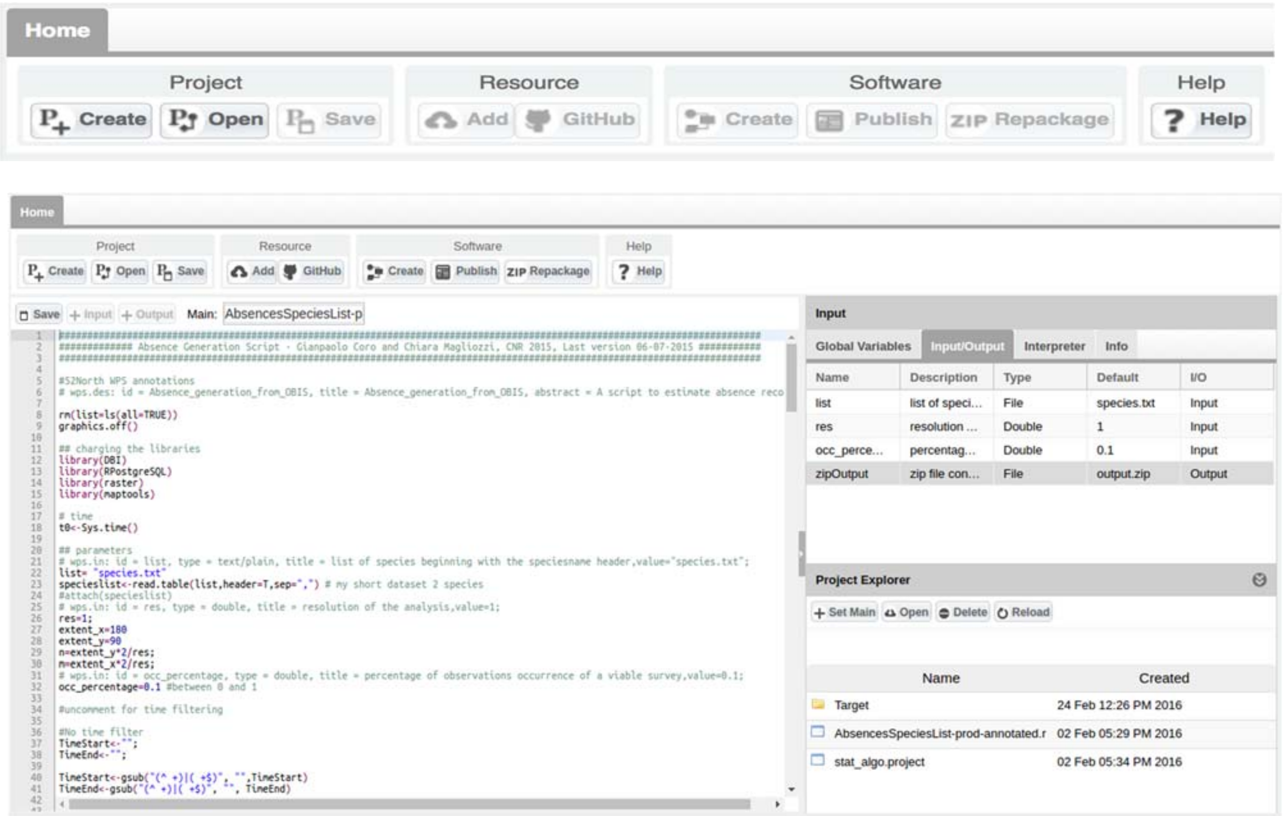


Figure 1. SAI Interface to import new methods in gCube-based data analytics engine.

The *Project* button allows creating, opening and saving a working session. A scientist uploads a set of files and data on the workspace area (lower-right panel). As a next step, the user indicates the "main script", i.e.

the script that will be executed on gCube-based data analytics engine and that will use the other scripts and files. After selecting the main script, the left-side editor panel visualises it with R syntax highlighting and allows modifying it. Subsequently, the user indicates the input and output of the script by highlighting variable definitions in the script and pressing the *+Input* (or *+Output*) button: behind the scenes the application parses the script strings and guesses the name, description, default value and type of the variable. This information is visualised in the top-right side *Input/Output* panel, where the user can modify the guessed information. As another option, SAI can automatically compile the same information based on WPS4R[1] annotations in the script. Other tabs in this interface area let setting global variables and adding metadata to the process. Specifically, the *Interpreter* tab allows indicating the R interpreter version and the packages required by the script and the *Info* tab allows declaring the name of the algorithm and its description while in the *Info* tab the user can set up the VRE in which the method should be available.

Once the metadata and the variables information have been compiled, the user can create a gCube-based data analytics engine as-a-Service version of the script by pressing the *Create* button in the Software panel. The term "software", in this case means a Java program that implements an as-a-Service version of the user-provided scripts. The Java software contains instructions to automatically download the scripts and the other required resources on the server that will execute it, configure the environment, execute the main script and return the result to the user. The computations are managed by the gCube-based data analytics engine computing platform that guarantees the program has one instance for each request and user. The servers will manage concurrent requests from several users and execute code in a closed sandbox folder, to avoid damage caused by malicious code.

To recapitulate, the SAI Web application enables Java, Python or R scripts with as-a-Service features and reduces integration time with respect to direct Java code writing so as to ease the **full integration** pattern which was reported to be a but complex in the previous experiences.

Once fully integrated, the application actually becomes a set of SoBigData social mining assets that will benefit from:

- **Scalability** Will benefit from a distributed and scalable computing platform;
- **Repurposing** Can be exploited in the context of many virtual research environments and it is suitable for being repurposed / applied to datasets;
- **Standard accessibility** Will be automatically made available via a web-based GUI as well as with web-based protocols (SOAP and Rest);
- **Accounting** Are monitored and assessed by SoBigData tools, e.g. detailed statistics on usage are transparently collected.

[1] 52North. WPS4R. https://wiki.52north.org/Geostatistics/WPS4R

4 RESOURCE INTEGRATION USE-CASES

This section reports on the experiences of scientists, as described by the scientists themselves, in integrating their applications, methods, and services in the infrastructure, including those cases where the integration is ongoing. Experiences are organized by typology of resources to be integrated and each of them reports on the effort required/envisaged, in terms of time and technical skills required for the integration, describing the high-level procedure that was followed and the highlighting the gaps for feedback and further improvement.

4.1 APPLICATIONS INTEGRATION EXPERIENCES

4.1.1 TAGME (MARCO CORNOLTI, UNIPI)

TagMe is an entity linking annotator, namely a software that, given a textual document, links mentions of entities found in the document towards a catalogue of entities drawn from a knowledge base. This has the important effect of building, on top of the document, a non-ambiguous representation of the topics mentioned by it, with impact on NLP applications such as information extraction, question answering, document topical clustering and categorization.

TagMe is implemented in Java (hence its execution requires a Java Virtual Machine) and its functionalities are offered as a web service for easy interoperability.

The integration in the D4Science infrastructure required the deployment of an ad-hoc virtual machine, running a SmartGears distribution. SmartGears came with range of additional services, including account registration, authentication and usage accounting with minimal effort. As a middleware to deploy the Tagme webapp, we employed the Tomcat web server, that had to be configured for the specific application in order to not let requests overload the server thereby rendering service unavailable. We also had to build a specific Tomcat Valve in order to have the TagMe webapp accept UTF8-encoded text as input, since the default charset did not cover the whole unicode space. We setup a web page presenting the service, the documentation, and an online demo of the service available to unauthenticated users. We took the occasion of the deployment of TagMe to rebuild its indexes according to the latest versions of Wikipedia, and fix a few minor bugs.

TagMe was previously available in a different deployment, and users had to be migrated to the new endpoint and authentication mechanism. We decided to migrate users in "waves", in order to progressively solve potential problems. The migration was concluded in September 2016, when the former deployment was shut down.

Deploying the service on a specific D4Science VRE had an important, unexpected advantage: the VRE message board became a forum for TagMe users and a mean of cooperation for research and problem solving, and an important channel of communication between users and TagMe administrators/developers. The deployment of a prototype of Tagme, with the only aim of testing, took two weeks and was ready for mid-May 2016. Until June, we fixed issues (often reported by users) with the web interface and the server configuration. The service was launched on mid-July, when the first wave of 1% of active users was migrated to the new platform, and the registration to the platform opened to the public. The service launch was finalized at the end of August, when all users were migrated to the new platform and the former service shut down.

In total, the deployment of Tagme roughly took four months.

The TagMe VRE is accessible here.

4.1.2 SMAPH (MARCO CORNOLTI, UNIPI)

SMAPH (paper) is an entity linking annotator built specifically for queries, a very peculiar type of textual documents, since they usually provide limited context, no grammaticality, and feature typos. SMAPH is the state of the art for entity linking in queries, and obtained the highest score in the ERD Challenge 2014. It is built on top of search engines (currently Google Search), and uses their output to annotate a query. It also makes use of statistical machine learning to make its decisions. Its impact on queries is similar to the one TagMe has on longer text, in that the semantic layer it builds on top of queries makes it possible to go deeper (with respect to syntactic text analysis) into the user need behind a query.

From the architectural point of view, Smaph is similar to TagMe in certain aspects: it is a Java software available through a web service. Similarly to TagMe, it has been deployed on an ad-hoc virtual machine running a SmartGears distribution. The main difference with respect to TagMe is that it needs access to the Google Custom Search API, hence users have to provide a Google authentication token to SMAPH when issuing a disambiguation request.

The deployment of the SMAPH web service was easier than that of TagMe, because no users had to be migrated, since this was the first time is was deployed. The deployment process started in mid-November and the service was released in mid-December, hence it took one month.

The SMAPH VRE is accessible here.

4.1.3 TWITTER MONITOR (STEFANO CRESCI AND SALVATORE MINUTOLI, IIT-CNR)

The Twitter Monitor is an interactive Web application designed to access and to collect data from the Twitter stream, by exploiting the public Twitter Streaming APIs. The application is able to manage concurrent monitors: it is possible to launch parallel listening sessions (i.e., more than one Twitter crawler at the same time) using different parameters and collecting different sets of data. In addition to offering an interactive Web interface in order to ease all the operations related to Twitter crawling, the Twitter Monitor also offers a set of functionalities aimed at minimizing the loss of data due to network or local machine problems. The Twitter Monitor is automatically capable of detecting and recovering from simple error situations, such as a closed or disconnected Twitter stream. It is also capable of detecting more serious issues, such as Twitter refusing to open new streaming connections, and automatically sends targeted alerts to system administrators.

The integration activities related to the Twitter Monitor have been twofold. On the one hand, we decided to take a course of action so as to provide a first, loosely-integrated, version of the Twitter Monitor as fast as possible. On the other hand, we also immediately started a deeper, yet longer, integration procedure.

The fast-lightweight integration allowed us to register the Twitter Monitor service in the SoBigData catalogue[2] and to include a navigation tab inside the SoBigData VRE pointing to an iframe that contains the first version of the SoBigData Twitter Monitor[3]. The SoBigData Twitter Monitor rendered inside the iframe is a slightly modified instance of the standalone monolithic Twitter Monitor that has been developed by IIT-CNR. It is a single PHP application that runs on an IIT-CNR virtual machine. Given this lightweight integration, the Twitter monitor is able to provide its services to the SoBigData users, but it cannot scale to serve a large number of requests. The implementation effort required to modify the original Twitter Monitor so that it could be used from inside the SoBigData VRE was about 2 weeks of work. The result of this integration is visible in Figure 2 and Figure 3 and has been extensively described in Deliverable D8.2 Crowdsensing Platform.

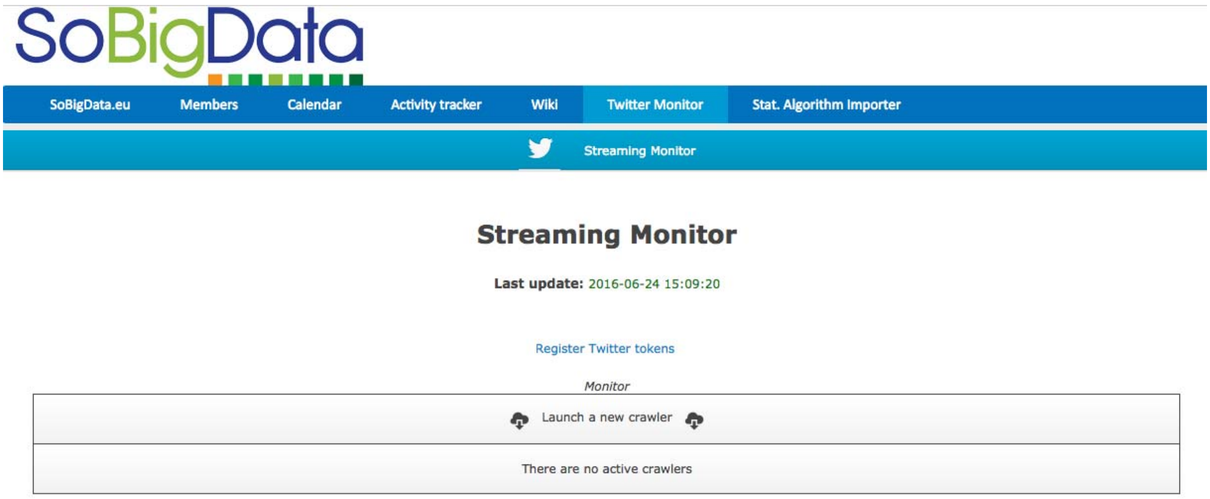


Figure 2. Twitter Monitor application rendered inside a dedicated navigation tab from the SoBigData VRE.

[2] https://sobigdata.d4science.org/group/resourcecatalogue/data-catalogue

[3] https://sobigdata.d4science.org/group/sobigdata.eu/twitter-monitor

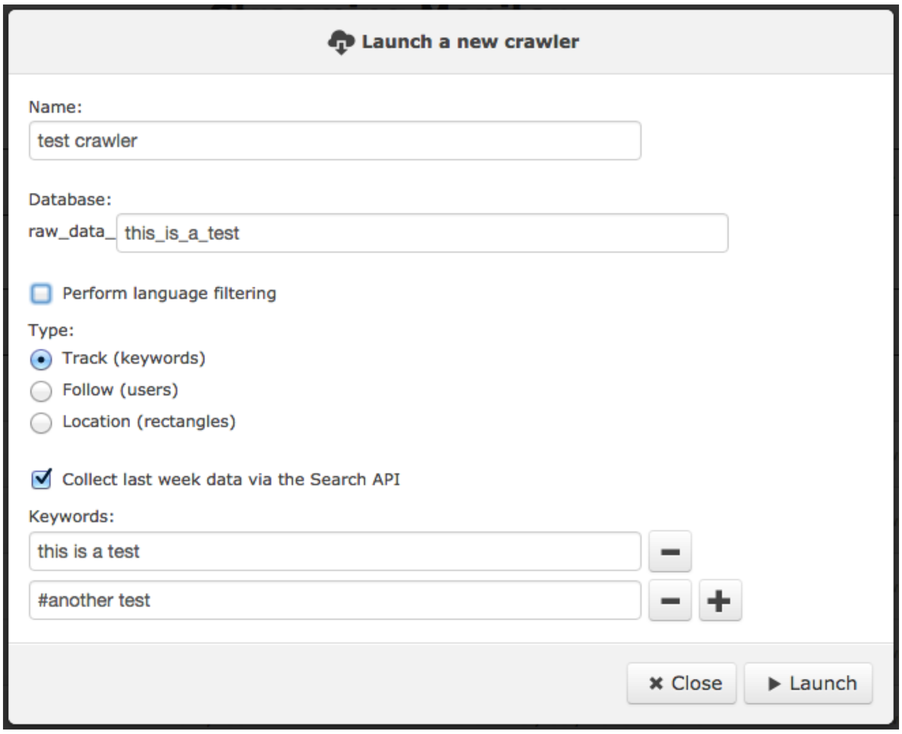


Figure 3. Twitter Monitor dialog window to allow parameter configuration of a new crawler.

While completing the lightweight integration, we also started the design and implementation of the full integration. At first, we had to reengineer and refactor the original monolithic Twitter Monitor application. Specifically, we split the whole application into 3 distinct logical modules, so as to allow a dynamic and scalable deployment of needed modules to the SoBigData eInfrastructure nodes. Then, we had to reimplement in Java the functionalities of the original PHP modules.

The new, fully-integrated, TwitterMonitor is composed of three modules: the Scheduler, the Cron and the Crawler. These modules were originally implemented in PHP. In order to integrate their functionalities inside the SoBigData infrastructure the Scheduler and the Cron have been replaced by equivalent Java modules. For the Crawler, due to its higher complexity, we decided to implement a *SmartExecutor* plugin that wraps the original PHP module by running it as a process. The Scheduler provides a GUI to let the user specify some input parameters needed to filter the Twitter events to gather. It has been implemented as a *StandardLocalExternalAlgorithm* subclass. The Cron runs at fixed intervals without user interaction and has been implemented as a *SmartExecutor* plugin. The Crawler is run/stopped by the Cron and has been implemented as a *SmartExecutor* plugin. The modules interact with each other by means of a database that contains the list of tasks to run and their status. As a first step, we decided to setup a *SmartGears* node to host the *SmartExecutor* plugins: this helped managing the plugin deploying more easily, without involving other system administrators. In the same virtual machine hosting the *SmartGears* node we deployed the postgres database, needed by the plugins, and registered it within the SoBigData infrastructure. The *SmartGears* installation simply required running a setup script. Then after few configuration steps it was ready to run. We also installed the *SmartExecutor* service to make the plugins accessible within the eInfrastructure. We implemented the modules by using some templates available in the documentation. Some support from developers helped us in speeding up the implementation.

The implementation of modules required a review of the functionalities of the infrastructure and, in particular, its interaction with the modules (i.e. the plugins). The main concerns were about discovering the reference for the database, interacting with the user, storing data in the user workspace, finding and running other plugins. During this activity, the D4Science developer team supported us by providing code snippets and quick links to the proper documentation.

4.1.4 WAT (MARCO PONZA, UNIPI)

WAT deprecates **TagME**: it has similar runtime performance but the result is more accurate (see the related paper [4] for details), though it can currently process English and well-formed (news, Web pages and RSS feeds) documents only. WAT is also significantly faster than TagMe for long document, which make it the perfect choice for an annotation at Web-scale of large document collections.

Since the WAT service deprecates TagME, it was implemented in Scala (hence its execution requires a Java Virtual Machine) and its functionalities are offered as a web service for easy interoperability it was decided to deploy it in the TagME VRE. The integration with the SoBigData infrastructure has been fully achieved in roughly 12 months and required the deployment of an ad-hoc virtual machine, running a SmartGears distribution. As a middleware to deploy the WAT webapp, we employed the Tomcat web server properly configured for the WAT application in order fully exploit the additional services SmartGears provides with minimal effort (account registration, authentication and usage accounting).

WAT's API is very similar to TagMe, thus ease a possible migration from the older entity linker. The documentation of WAT's public API is provided at https://services.d4science.org/web/tagme/wat-api

Deploying the service on a the TagMe VRE had an important advantage: like for the TagMe case the VRE message board became a forum for WAT users and a mean of cooperation for research and problem solving, and an important channel of communication between users and WAT administrators/developers. The integration of WAT was concluded in January 2019.

The TagMe VRE is accessible here.

4.1.5 SWAT (MARCO PONZA, UNIPI)

SWAT is an **entity-salience** system which identifies on-the-fly the **semantic focus** of a document, expressed by its Salient Wikipedia Entities. The core of this technology is based on a broad set of syntactic and semantic features, extracted from the input document and later fed to a classifier, previously trained on millions of training examples extracted from the New York Times Annotated Corpus.

The core of SWAT is written in Python, but it functionalities are actually built on the top of WAT (cf. 4.1.4). SWAT's VRE queries WAT through a RESTful API and then it uses these responses for its entity-salience computation. A similar, but simpler, web interface is provided by SWAT to external web calls in order to ease its use in other software tools. Since SWAT exploits WAT it was decided to deploy it in the TagME VRE.

[4] https://dl.acm.org/citation.cfm?id=2634350

Being the core of SWAT implemented in python the integration in the SoBigData infrastructure required not only the deployment of an ad-hoc virtual machine, running a SmartGears distribution (which would bring a range of additional services, including account registration, authentication and usage accounting with minimal effort). Rather. we implemented a reverse-proxy component in Java, deployed on the same SmartGears container, which would receive the requests and then forwards them to the SWAT (python) internal handler.

An experimental GUI of the system is available at https://swat.d4science.org/ and the documentation of its public API is available at https://services.d4science.org/web/tagme/swat-api. The integration of SWAT in the SoBigData infrastructure is now completed with Accounting e AuthZ functionalities, it was concluded in January 2019.

The TagMe VRE is accessible here.

4.2 METHODS INTEGRATION EXPERIENCES

4.2.1 TRAJECTORYBUILDER (ROBERTO TRASARTI, ISTI-CNR)

Trajectory Builder (https://ckan-sobigdata.d4science.org/dataset/trajectory_builder) is a basic algorithm of the M-Atlas package (http://m-atlas.eu/). It is implemented in Java and interact with a Postgres database (with Postgis extension for spatial primitives) in order to build a trajectory from raw spatio-temporal observations, i.e. points in space and time represented by a triple <latitude, longitude, timestamp>. Hence the dependencies of this algorithm are: the M-Atlas cose (matlas.jar) and the connector drivers for postgresql (postgresql-8.4-701.jdbc4.jar).

In order to be integrated a launcher class is implemented as a subclass of StandardLocalExternalAlgorithm (https://sobigdata.d4science.org/group/sobigdatalab/importer-documentation) . The Main problem during the integration was the configuration of the project into Maven for a deploy inside the SoBigData platform due the dependencies listed above. Thanks to the platform administrator help the problems were solved and the deploy was done. The time passed from the beginning of the integration and the publish of the algorithm in the VRE Lab is 1 month, but the actual work in implementing the tool was only few hours, the rest of the time was spent interacting (1-2 times each week) with the platform administrators and developers in order to find the right solution for the integration and how to put the dependencies in it. The difficulties of the process of integration with the SoBigData platform engine are:

- The documentation is so wide that is difficult to understand what to do and why.
- It is required know tools such as Maven and how to configure it to be compliant with the platform.
- The interaction with the platform developer sometimes was difficult trough the ticketing system, face to face meeting were fundamental to solve the problems.

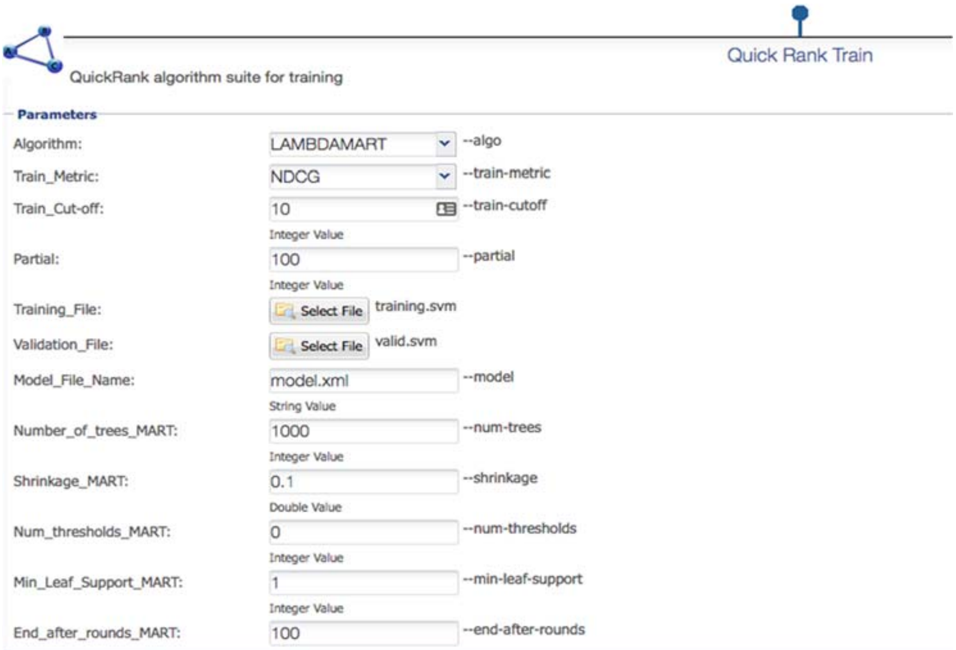
Anyway several of the problems solved during this first integration will help to be more independent in the integration of next algorithm.

4.2.2 QUICKRANK (CRISTINA MUNTEAN, ISTI-CNR)

QuickRank[5] is an efficient Learning-to-Rank toolkit providing several C++ implementation of LtR algorithms. The LtR algorithms currently implemented are: GBRT, LamdaMART, Oblivious GBRT / LamdaMART, CoordinateAscent, LineSearch, RankBoost.

We made QuickRank available to SoBigData by integrating it in the SoBigData Lab VRE as a WPS method executable from VRE Method Engine[6]. The information about the integrated tool can be found in the SoBigData Catalogue at the following link: https://ckan-sobigdata.d4science.org/dataset/quickrank .

QuickRank users can train and test models with the help of a Command Line Interface (CLI). In order to integrate our tool in the SoBigData VRE we had to create a WPS wrapper around it, allowing the user to interact with it in a more user-friendly way, through the help of an User Interface (UI). He/she can insert the files and options as he invokes the training or testing capabilities on the existing LtR algorithms as he would do with a CLI. Figure 4 and Figure 5 show the interface for the training and test tools/interfaces created.



[5] http://quickrank.isti.cnr.it/

[6] https://services.d4science.org/group/sobigdatalab/method-engine

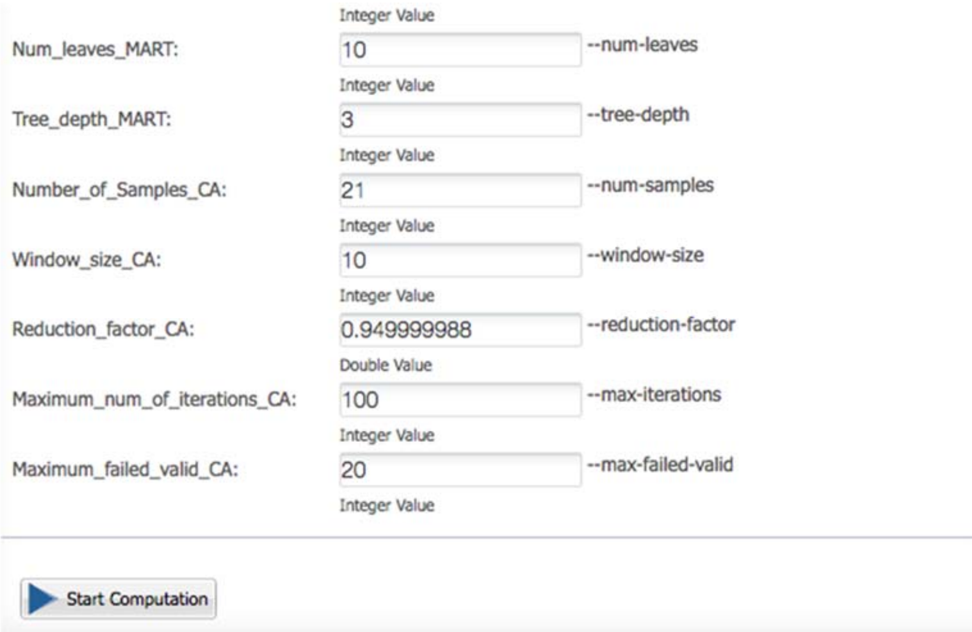


Figure 4. QuickTest training user interface

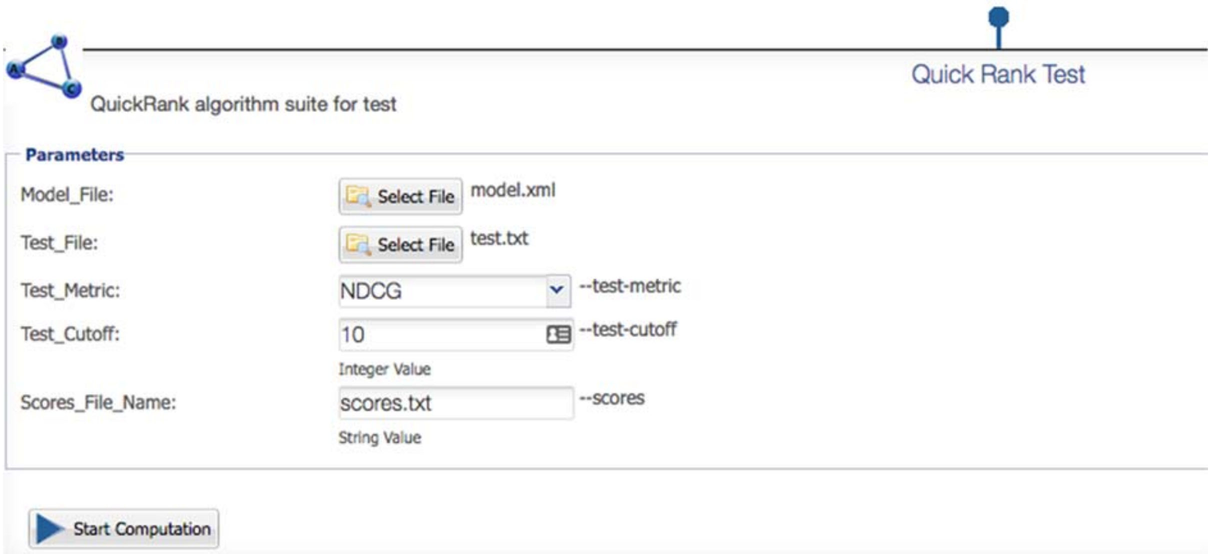


Figure 5. QuickRank test user interface

The QuickRank WPS interfaces were implemented in Java, by extending the *StandardLocalExternalAlgorithm*[7] class, put at our disposal by the team responsible with the integration. In order for the QuickRank wrapper to work, several libraries needed to be installed on the VRE servers. QuickRank needs gcc 4.9 (or above), CMake 2.8 (or above) and git. The implementation effort required to create a wrapper around a C++ executable, with the help of the *Process Builder* class , which allows us to

[7] As described in: https://wiki.gcube-system.org/gcube/How-to_Implement_Algorithms_for_the_Statistical_Manager

run the QuickRank executable from the Java class, while also passing all the necessary parameters as provided in input by the user in the UI.

The whole integration effort lasted several months. The task was started on the 15th of February 2016, and was the first tool to be integrated in the SoBigData VRE. Initially, we developed a test wrapper on the development servers, this phase ending on the 22nd of March. Later, on the 16th of August, the second phase started, when the actual wrappers were inserted and deployed into the production environment in the SoBigData VRE. The integration effort ended onl the 30th of September.

We integrated the two main modes (3 wrappers in total) in which QuickRank can be used:

- Training with validation set
- Training without validation set
- Test

The results/outputs of the QR computation can be seen in the Output Datasets section (Figure 6)

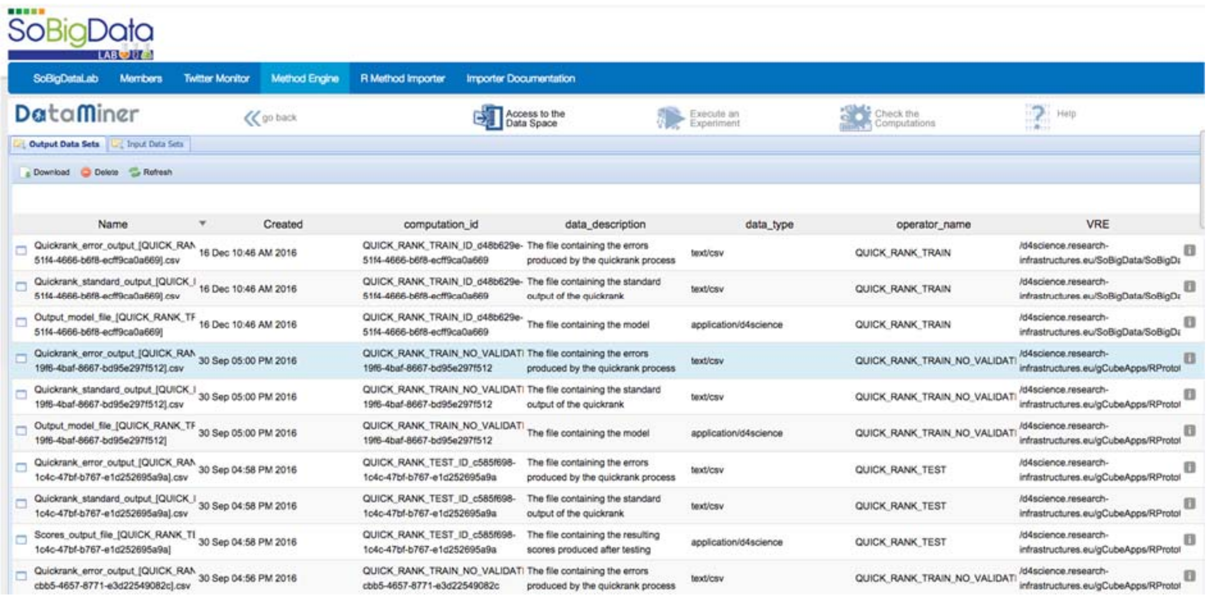


Figure 6. QuickRank Output file

The easiest part was writing the wrapper in Java, namely extending the *StandardLocalExternalAlgorithm* class. With the help of the integration team we were able to overcome some of the issues (e.g. identify the proper data types in the gCube wiki) create the proper interfaces, fix all the problematic issues and deploy the project locally. In order to be put in production QuickRank need to have the dependencies installed and the source compiled. For this we opened a ticket, which took around 2 weeks to be resolved. Soon after that we succeeded in finalizing the integration and test the algorithms also on the production server.

4.2.3 GATECLOUD (KALINA BONTCHEVA, UNIVERSITY OF SHEFFIELD)

SoBigData users will access the GATE Cloud service indirectly through the D4Science platform. GATE Cloud is today deployed at Sheffield and exposes calls to methods as REST services. Such methods will be integrated as WPS method calls from the e-infrastructure environments. More specifically, SoBigData will mediate end-user requests (invocations of a service) to GATE Cloud by using a special API key, which will bypass the accounting inherent in GATE Cloud and use that provided by the VRE uniformly instead.

Around 23 methods available as GateCloud REST services have been exposed through the SoBigData VRE in the last period, each with their own endpoint. The API for each will be identical, accepting as input documents in XML, JSON, plain text or HTML format, producing output as JSON, HTML, XML.

Since the end of December 2018 we were able to integrate other 34 new GateCloud REST services, with the same technique exploited for the previous 23, for a total number of 57 services as shown in Figure 7. The time passed from the beginning of the integration and the publishing of the new 34 algorithms in the VRE Lab was 3 months.

These methods will be listed in VRE catalogue and be made accessible via a Method Engine application. Eventually, the VRE catalogue and the GATE Cloud catalogue will be synchronised, so that VRE users can discover methods from GATE Cloud directly. This will be supported by the HTTP API currently part of the GATE Cloud service. These services may be grouped into a single Data Miner instance for ease of discovery.

In the long term, GATE Cloud will be adapted for deployment on the SoBigData hardware premises, allowing cost-free access to GATE Cloud services by researchers within the scope of VREs.

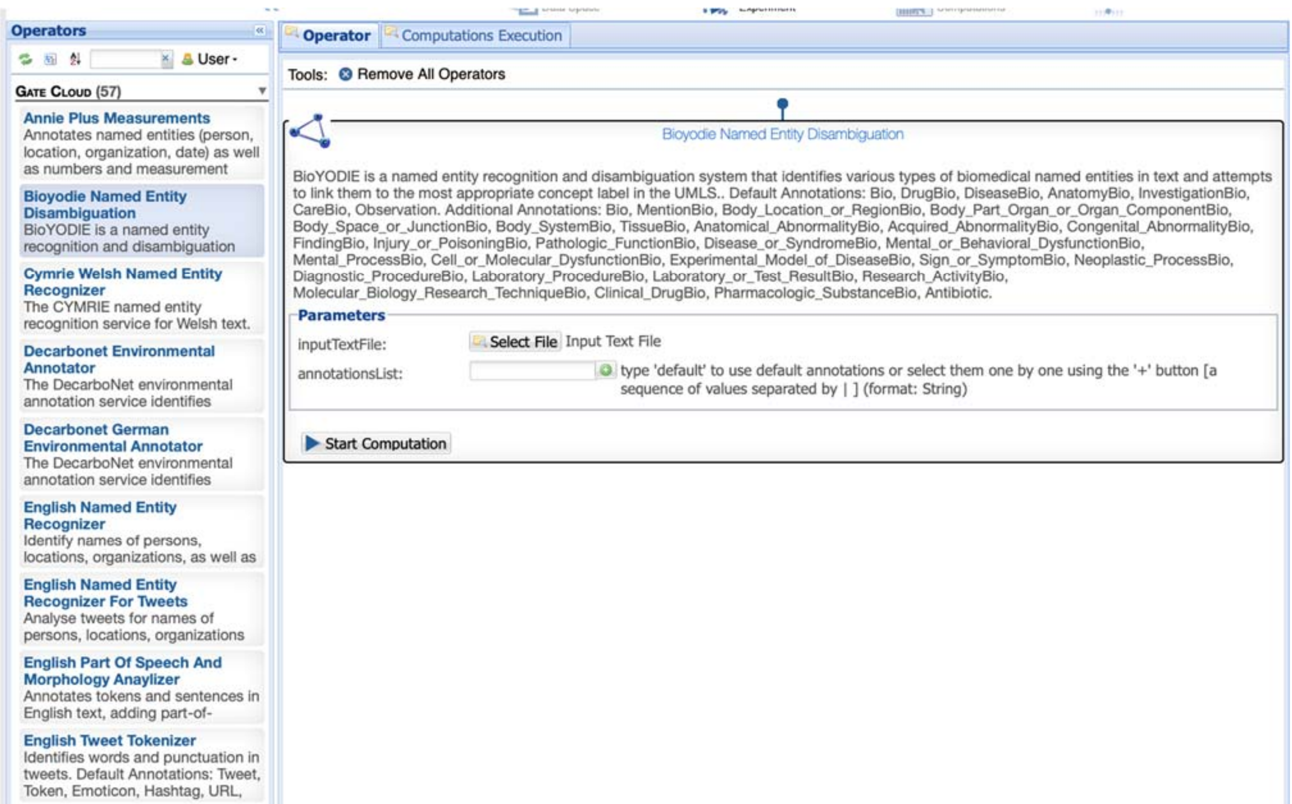


Figure 7. Web interface of the GATE Cloud methods

4.2.4 STATVAL (DANIELE REGOLI, SNS)

StatVal is an algorithm performing a statistical validation filter for complex networks.

The purpose of StatVal is taking a (possibly large and dense) complex network and find the links that are not explainable by simple random wiring of the nodes given the degree/strength of nodes (namely, given how many links each node has attached). The result is a new network with only the links that are unexpected with respect to the said random model. Thus, you come up with an output network much sparser than the original, that can be considered a statistically sound representation of the network backbone structure.

StatVal is essentialy a collection of R scripts. The integration has been performed through the *Statistical Algorithm Importer* (SAI) available in the D4Science platform, which supports tools for the easy integration of R script methods inside a VRE. StatVal is now available for use in the *SoBigDataLab* environment via a Method Engine application (see Figure 8). You can find it on the resource catalogue.

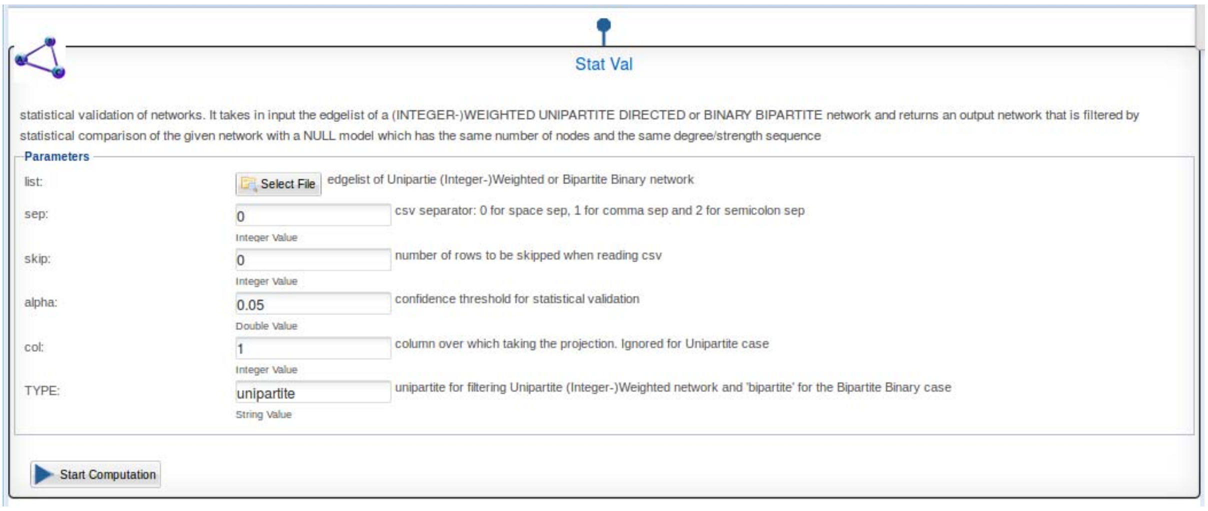


Figure 8. Web interface of the StatVal method

The user has simply to load a network in the form of a csv edgelist and tune some parameters either related to the input network or to the confidence level of the statistical tests and run the method. Output is a compressed archive containing edgelists of (different) filtered networks.

Usage of SAI for R scripts integration is quite straightforward. First we needed to create a project folder, then we simply had to load in the folder the necessary R scripts. A main R script, which is going to manage the workflow of the algorithm, is to be given to the SAI. Finally, we declared the inputs and outputs of the algorithm and what R packages the algorithm requires.

Some more effort was needed in the testing part, namely when checking whether the method was working properly, and in the maintenance, namely in doing changes and modifications to the algorithms, mainly due to some misunderstandings and some problems in using test data stored inside the same folder of the software project. But with the help of D4Science administrators all problems were solved.

Total effort, counting both the first integration and successive modifications, was roughly around a week full time.

4.2.5 OPTICS CLUSTERING (ROBERTO TRASARTI, ISTI-CNR)

Optics Clustering is one of the tools provided by M-Atlas[8] and is implemented in Java to interact with a Postgres database (with Postgis extension for spatial primitives) to group similar trajectories using the well known Optics algorithm [9]. The implementation includes a set of similarity functions specialized in trajectory data and mobility applications. The dependencies are the M-Atlas code (matlas.jar) and the connector drivers for postgresql (postgresql-8.4-701.jdbc4.jar). In order to be integrated a launcher class is

[8] M-Atlas is a mobility querying and data mining system centered onto the concept of spatio-temporal data. Besides the mechanisms for storing and querying trajectory data, M-Atlas has mechanisms for mining trajectory patterns and models that, in turn, can be stored and queried.
[9] Ankerst, Mihael & M. Breunig, Markus & Kriegel, Hans-Peter & Sander, Joerg. (1999). OPTICS: Ordering Points to Identify the Clustering Structure. Sigmod Record. 28. 49-60. 10.1145/304182.304187.

implemented as a subclass of StandardLocalExternalAlgorithm (https://sobigdata.d4science.org/group/sobigdatalab/importer-documentation). The implementation followed was very easy thanks to the experience with the previous integration (i.e. Trajectory Builder).

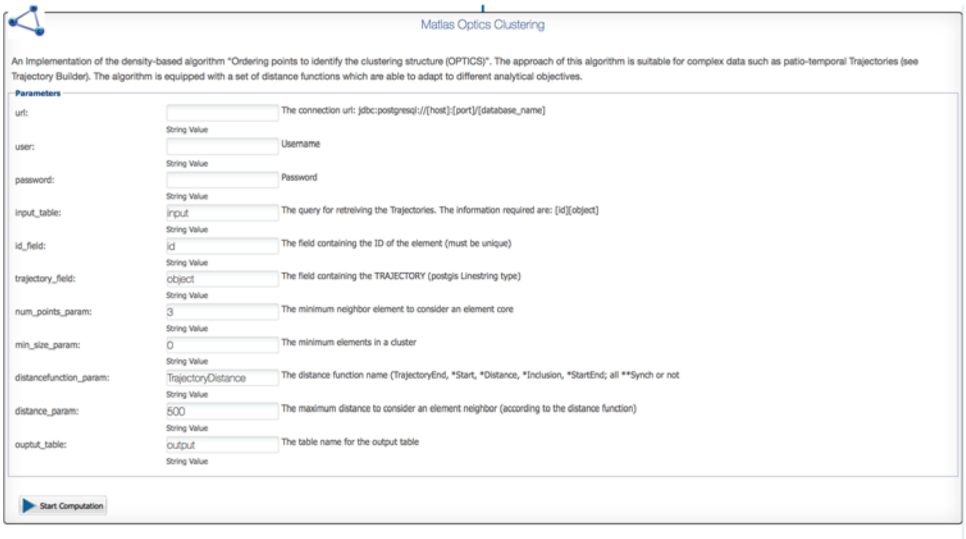


Figure 9. Web interface of the Optics Clustering method

The configuration of the project into Maven for the deployment was the major obstacle due the lack of true understanding of some mandatory steps which can be done only by following the expert guidance. The time passed from the beginning of the integration and the publishing of the algorithm in the VRE Lab was 2 weeks.

4.2.6 S-CUBE: A FRAMEWORK FOR DISCOVERING SEGREGATION IN BIPARTITE NETWORKS (ALESSANDRO BARONI AND SALVATORE RUGGIERI, UNIPI)

S-Cube is a is a segregation discovery method from both tabular and social network data. Segregation discovery consists of finding contexts of segregation of social groups distributed across units or communities. The S-Cube method implements an approach for segregation discovery on top of frequent itemset mining, by offering to the analyst a multi-dimensional (segregation) data cube for exploratory data analysis. Total effort for the integration was roughly around a month. A documentation page is also available at https://goo.gl/n59tve

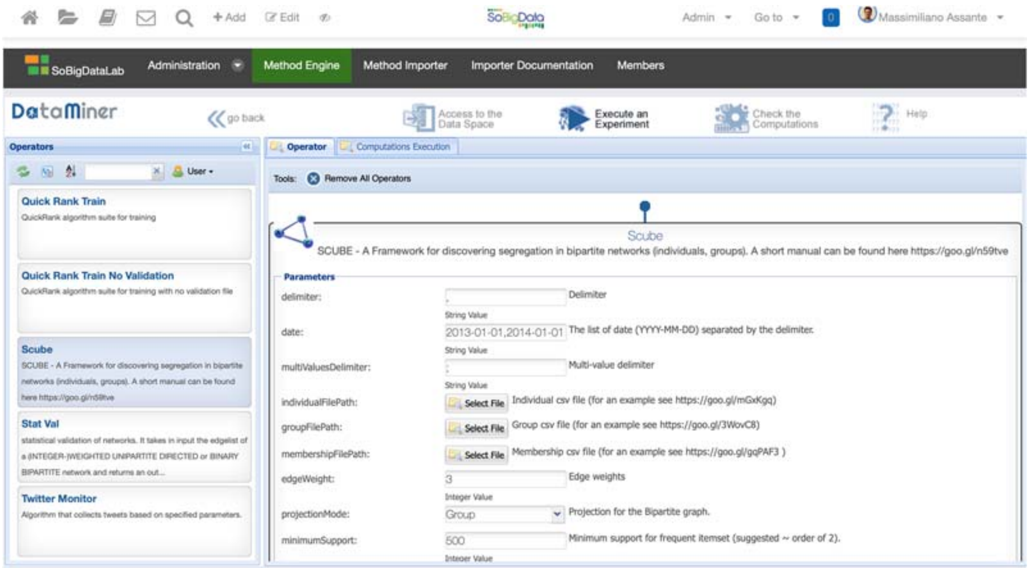


Figure 10. Web interface of the SCube method