

# Cloud computing for enabling Big Data analysis

Loris Belcastro<sup>1</sup>[0000-0001-6324-8108], Fabrizio Marozzo<sup>1,2</sup>[0000-0001-7887-1314],  
Domenico Talia<sup>1,2</sup>[0000-0003-1910-9236], and Paolo Trunfio<sup>1</sup>[0000-0002-5076-6544]

<sup>1</sup> DIMES, University of Calabria, Italy

<sup>2</sup> DtoK Lab Srl, Rende (CS), Italy

{lbelcastro,fmarozzo,trunfio,talia}@dimes.unical.it

**Abstract.** Every day millions of people use social media and produce huge amount of digital data that can be effectively exploited to extract valuable information concerning human dynamics and behaviors. Such data, commonly referred as Big Data, contains valuable information about user activities, interests, and behaviors, which makes it intrinsically suited to a very large set of applications. For getting valuable information and knowledge from such data in a reasonable time, novel frameworks and data analysis approaches have been developed. This paper aims at presenting some recent Cloud-based frameworks and methodologies for Big Data processing that can be used for developing and executing several data analysis applications, including trajectory mining and sentiment analysis. The paper is organized in two main parts. The first part focuses on tools for developing and executing scalable data analysis applications on Cloud. The second part presents data analysis methodologies for extracting knowledge from large datasets.

**Keywords:** cloud computing, big data analysis, sentiment analysis, trajectory mining, social data analysis, sentiment analysis

## 1 Introduction

In the last years the ability to produce and gather data has increased exponentially. In the Internet of Things' era, huge amounts of digital data are generated by and collected from several sources, such as sensors, mobile devices, web applications and services. Moreover, with the widespread diffusion of mobile devices, every day millions of people use social media and produce huge amount of digital data that can be effectively exploited to extract valuable information concerning human dynamics and behaviors. Such data, commonly referred as Big Data, contains valuable information about user activities, interests, and behaviors, which makes it intrinsically suited to a very large set of applications [6]. The huge amount of data generated, the speed at which it is produced, and its heterogeneity in terms of format, represent a challenge to the current storage, process and analysis capabilities. Then to extract value from such kind of data, novel frameworks and data analysis approaches have been developed for capturing and analyzing complex and/or high velocity data.

In this scenario, high performance computers, such as many and multi-core systems, Clouds, and multi-clusters, paired with parallel and distributed algorithms are commonly used by data analysts to tackle Big Data issues and get valuable information and knowledge in a reasonable time. In particular, Cloud computing systems provide large-scale computing infrastructures for complex high-performance applications, such as those that use advanced data analytics techniques for extracting useful information from large, complex datasets [29]. However, combining Big Data analytics techniques with scalable computing systems will produce new insights in a shorter time. Although a few Cloud-based analytics platforms are available today, current research work foresees that they will become common within a few years.

This paper aims at presenting some recent Cloud-based frameworks and methodologies for Big Data processing that can be used for developing and executing several data analysis applications, including trajectory mining and sentiment analysis. The paper is organized in two main parts. The first part focuses on tools for developing and executing scalable data analysis applications on Cloud. The second part presents data analysis methodologies for extracting knowledge from large datasets.

In particular, the paper is organized as follows. Section 2 presents a Data Mining Cloud Framework designed for developing and executing distributed data analytics applications as workflows of services. In such environment data sets, analysis tools, data mining algorithms and knowledge models are implemented as single services that can be combined through a visual programming interface in distributed workflows to be executed on Clouds. Section 3 describes a high-level library for developing parallel data mining applications based on the extraction of useful knowledge from large dataset gathered from social media. The library aims at reducing the programming skills needed for implementing scalable social data analysis applications.

Section 4 presents a Software-as-a-Service (SaaS) system that exploits Cloud facilities to provide efficient services for analyzing large datasets. The system allows users to import their data to the Cloud, extract knowledge models using high performance data mining services, and exploit the inferred knowledge to predict new data and behaviors. Section 5 describes SMA4TD, a methodology for discovering behavior and mobility patterns of users attending large-scale public events, by analyzing social media posts.

Section 6 presents novel Region-of-Interest (RoI) mining technique that exploits the indications contained in geotagged social media items (e.g. tweets, posts, photos or videos with geospatial information) to discover RoIs with high accuracy. Section 7 presents a methodology for discovering the polarization of social media users during election events characterized by the competition of political factions. The methodology uses an automatic incremental procedure based on neural networks for analyzing the posts published by social media users.

Finally, Section 8 concludes the paper.

## 2 Data Mining Cloud Framework (DMCF)

The Data Mining Cloud Framework (DMCF) [24] is a software system for designing and executing data analysis workflows on Clouds. DMCF supports a large variety of data analysis processes, including single-task applications, parameter sweeping applications [23], and workflow-based applications. A Web-based user interface allows users to compose their applications and to submit them for execution to a Cloud platform, according to a Software-as-a-Service approach.

The DMCF's architecture has been designed to be implemented on different Cloud systems, so as to take advantage of main cloud computing features, such as elasticity of resources provisioning. In DMCF, at least one Virtual Web Server runs continuously in the Cloud, as it serves as user front-end. In addition, users specify the minimum and maximum number of Virtual Compute Servers, which are in charge of executing the data mining tasks. The DMCF can exploit the auto-scaling features that allows dynamic spinning up or shutting down Virtual Compute Servers, based on the number of tasks ready for execution in the DMCF's Task Queue. Since storage is managed by the Cloud platform, the number of storage servers is transparent to the user.

### 2.1 Workflow formalisms

The DMCF allows creating data mining and knowledge discovery applications using workflow formalisms. Workflows may encompass all the steps of discovery based on the execution of complex algorithms and the access and analysis of scientific data. In data-driven discovery processes, knowledge discovery workflows can produce results that can confirm real experiments or provide insights that cannot be achieved in laboratories. In particular, DMCF allows to program workflow applications using two languages:

1. *VL4Cloud* (Visual Language for Cloud), a visual programming language that lets users develop applications by programming the workflow components graphically [26].
2. *JS4Cloud* (JavaScript for Cloud), a scripting language for programming data analysis workflows based on JavaScript [25].

Both languages use two key programming abstractions:

1. *Data* elements denote input files or storage elements (e.g., a dataset to be analyzed) or output files or stored elements (e.g., a data mining model).
2. *Tool* elements denote algorithms, software tools or service performing any kind of operation that can be applied to a data element (data mining, filtering, partitioning, etc.).

In particular, three different types of Tools can be used in a DCMF workflow:

1. A *Batch Tool* is used to execute an algorithm or a software tool on a Virtual Compute Server without user interaction. All input parameters are passed as command-line arguments.

2. A *Web Service Tool* is used to insert into a workflow a Web service invocation.
3. A *MapReduce Tool* is used to insert into a workflow the execution of a MapReduce algorithm or application running on a cluster of virtual servers [7].

For each Tool in a workflow, a *Tool descriptor* includes a reference to its executable, the required libraries, and the list of input and output parameters. Each parameter is characterized by *name*, *description*, *type*, and can be *mandatory* or *optional*. As an example, a MapReduce Tool descriptor is composed by two groups of parameters: *generic parameters*, which are parameters used by the MapReduce runtime, and *applications parameters*, which are parameters associated to specific MapReduce applications. In the following, we list a few examples of generic parameters:

- *mapreduce.job.reduce*: the number of reduce tasks per job;
- *mapreduce.job.maps*: the number of map tasks per job;
- *mapreduce.input.fileinputformat.split.minsize*: the minimum size of chunk that map input should be split into;

Another common element is the task concept, which represents the unit of parallelism in our model. A *task* is a Tool, invoked in the workflow, which is intended to run in parallel with other tasks on a set of Cloud resources. According to this approach, VL4Cloud and JS4Cloud implement a *data-driven task parallelism*. This means that, as soon as a task does not depend on any other task in the same workflow, the runtime asynchronously spawns it to the first available virtual machine. A task  $T_j$  does not depend on a task  $T_i$  belonging to the same workflow (with  $i \neq j$ ), if  $T_j$  during its execution does not read any data element created by  $T_i$ .

In VL4Cloud, workflows are directed acyclic graphs whose nodes represent data and tools elements. The nodes can be connected with each other through direct edges, establishing specific dependency relationships among them. When an edge is being created between two nodes, a label is automatically attached to it representing the type of relationship between the two nodes. Data and Tool nodes can be added to the workflow singularly or in array form. A data array is an ordered collection of input/output data elements, while a tool array represents multiple instances of the same tool.

In early versions, DMCF has exploited the default storage provided by public cloud infrastructures for any I/O operations. This implies that DMCF's I/O performance was limited by the performance of the storage provided by cloud providers. In work [27] it was proposed to use the Hercules system within DMCF for storing temporary data generated by workflow-based applications. Hercules is a highly scalable, in-memory, distributed storage system [18]. In a later work [22], we also used a data-aware scheduling runtime that exploits data locality during the execution of workflows. An experimental evaluation was carried out to evaluate the advantages of these strategies and to demonstrate the effectiveness of the solution. Using the proposed data-aware strategy and Hercules as a temporary storage service, I/O overhead was reduced by 55% compared to standard Azure storage-based execution, leading to a 20% reduction in total execution of the workflow.

### 2.2 Workflow examples

Figure 1 shows an example of data analysis workflow developed using the visual workflow formalism of DMCF.

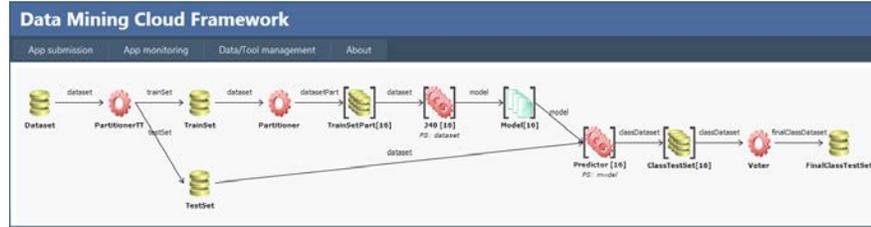


Fig. 1: Example of data analysis application designed using VL4Cloud.

In JS4Cloud, workflows are defined with a JavaScript code that interacts with Data and Tool elements through three functions:

1. *Data Access*, for accessing a Data element stored in the Cloud;
2. *Data Definition*, to define a new Data element that will be created at runtime as a result of a Tool execution;
3. *Tool Execution*, to invoke the execution of a Tool available in the Cloud.

Once the JS4Cloud workflow code has been submitted, an interpreter translates the workflow into a set of concurrent tasks by analysing the existing dependencies in the code. The main benefits of JS4Cloud are:

1. it extends the well-known JavaScript language while using only its basic functions (arrays, functions, loops);
2. it implements both a data-driven task parallelism that automatically spawns ready-to-run tasks to the Cloud resources, and data parallelism through an array-based formalism;
3. these two types of parallelism are exploited implicitly so that workflows can be programmed in a totally sequential way, which frees users from duties like work partitioning, synchronization and communication.

Figure 2 shows the script-based workflow version of the visual workflow shown in Figure 1. In this example, parallelism is exploited in the for loop at line 7, where up to 16 instances of the J48 classifier are executed in parallel on 16 different partitions of the training sets, and in the for loop at line 10, where up to 16 instances of the Predictor tool are executed in parallel to classify the test set using 16 different classification models.

Figure 2 shows a snapshot of the parallel classification workflow taken during its execution in the DMCF’s user interface. Beside each code line number, a colored circle indicates the status of execution. This feature allows user to monitor

```

1 var n = 16;
2 var DRef = Data.get("Dataset"), TrRef = Data.define("TrainSet"), TeRef = Data.define("TestSet");
3 PartitionerTT({dataset:DRef, percTrain:0.7, trainSet:TrRef, testSet:TeRef});
4 var PRef = Data.define("TrainsetPart", n);
5 Partitioner({dataset:TrRef, datasetPart:PRef});
6 var MRef = Data.define("Model", n);
7 for(var i=0; i<n; i++)
8   J48({dataset:PRef[i], model:MRef[i], confidence:0.1});
9 var CRef = Data.define("ClassTestSet", n);
10 for(var i=0; i<n; i++)
11   Predictor({dataset:TeRef, model:MRef[i], classDataset:CRef[i]});
12 var FRef = Data.define("FinalClassTestSet");
13 Voter({classData:CRef, finalClassData:FRef});

```

Fig. 2: Example of data analysis application designed using JS4Cloud.

the status of the workflow execution. Green circles at lines 3 and 5 indicate that the two partitioners have completed their execution; the blue circle at line 8 indicates that J48 tasks are still running; the orange circles at lines 11 and 13 indicate that the corresponding tasks are waiting to be executed.

### 2.3 Workflow study cases

DMCF has been used to implement several Big Data analytics applications, including a workflow for discovering patterns and rules from trajectory data [2]. Figure 3 shows the VL4Cloud workflow that define the steps of such application. Experimental evaluation has been carried out on GPS datasets tracing the movement of taxis in the urban area of Beijing. The results showed that, due to the high complexity and large volumes of data involved in the application scenario, the trajectory pattern mining process takes advantage from the scalable execution environment offered by DMCF in terms of both execution time, speed-up and scale-up.

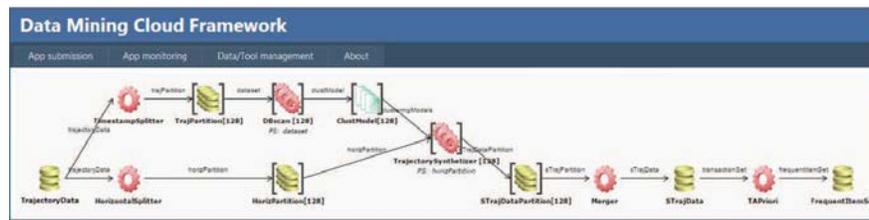


Fig. 3: Trajectories workflow composed and executed in the Data Mining Cloud Framework (DMCF).

DMCF has also been used to implement a Cloud-based computing infrastructure for the analysis of SNP microarray data [1]. It was possible to define a software tool (Cloud4SNP) for the parallel preprocessing and statistical analysis of pharmacogenomics SNP microarray data. Experimental evaluation shows

efficient execution times and very good scalability. Moreover, the system implementation shows how the exploitation of a Cloud platform allows researchers and professionals to face in an elastic way the requirements of small as well as very large pharmacogenomics studies.

DMCF also supports data classification workflows that include MapReduce computations. As an example, in [8] DMCF has been used to implement a MapReduce data analysis application for predicting flight delays. Every year approximately 20% of airline flights are delayed or canceled mainly due to bad weather, carrier equipment, or technical airport problems. The goal of this application is to implement a predictor of the arrival delay of scheduled flights due to weather conditions. To run the workflow, we used a Hadoop cluster composed of 1 head node and 8 worker nodes, over the cloud servers used by the DMCF environment. With this setting, the turnaround time decreased from about 7 h using 2 workers, to about 1.7 h using 8 workers, with a speedup that is very close to linear values.

### 3 Parallel Social Data Analysis (ParSoDA)

Several developers and researches are working on the design and implementation of tools and algorithms for extracting useful information from data gathered from social media. In most cases the amount of data to be analyzed is so big that high-performance computers, such as many and multi-core systems, Clouds, and multi-clusters, paired with parallel and distributed algorithms, are used by data analysts to reduce response time to a reasonable value [9]. Several research projects consider not only the data analysis task, but also procedures including other data processing tasks needed for building social data applications. In particular, these projects aim at helping scientists to implement all the steps that compose social data mining applications without the need to implement common operations from scratch.

ParSoDA (Parallel Social Data Analytics) [12] is a Java library that includes algorithms widely used to process and analyze data gathered from social media with the goal of extracting different kinds of information (e.g., user mobility, user sentiments, topic trends, and frequency). ParSoDA defines a general structure for a social data analysis application that is formed by the following steps:

- *Data acquisition*: during this step, it is possible to run multiple crawlers in parallel; the collected social media items are stored on a distributed file system (HDFS [28]).
- *Data filtering*: this step filters the social media items according to a set of filtering functions.
- *Data mapping*: this step transforms the information contained in each social media item by applying a set of map functions.
- *Data partitioning*: during this step, data is partitioned into shards by a primary key and then sorted by a secondary key.
- *Data reduction*: this step aggregates all the data contained in a shard according to the provided reduce function.

- *Data analysis*: this step analyzes data using a given data analysis function to extract the knowledge of interest.
- *Data visualization*: at this final step, a visualization function is applied on the data analysis results to present them in the desired format.

For each of these steps, ParSoDA provides a predefined set of functions. For example, for the data acquisition step, ParSoDA provides crawling functions for gathering data from some of the most popular social media platforms (e.g., Twitter and Flickr), while for the data filtering step, ParSoDA provides functions for filtering geotagged items based on their position, time of publication, and contained keywords. Users are free to extend this set of functions with their owns.

### 3.1 Reference architecture and execution flow

Figure 4 presents the reference architecture and execution flow of a ParSoDA application that runs on the Hadoop [30] or Spark [31] framework. In such way, it is possible to implement several parallel and distributed data mining applications with high scalability. As shown in Figure 4(a), user applications can utilize ParSoDA and other libraries (e.g., Mahout<sup>3</sup>, MLlib<sup>4</sup>). Applications can be executed on Hadoop or Spark, using YARN as resource manager and HDFS as distributed storage system. Figure 4(b) provides details on how applications are executed on a Hadoop or a Spark cluster. The cluster is formed by one or more master nodes, and multiple worker nodes. Once a user application is submitted to the cluster, its steps are executed according to their order (i.e., data acquisition, data filtering, etc.).

On a Hadoop cluster, some steps are inherently MapReduce-based, namely: *data filtering*, *data mapping*, *data partitioning* and *data reduction*. This means that all the functions used to perform these steps are executed within a MapReduce job that runs on a set of worker nodes. In particular, the data filtering and data mapping steps are wrapped within Hadoop Map tasks; the data partitioning step corresponds to Hadoop Split and Sort tasks; the data reduction step is executed as a Hadoop Reduce task. The remaining steps (data acquisition, data analysis, and data visualization) are not necessarily MapReduce-based. This means that the functions associated with these steps could be executed in parallel on multiple worker nodes, or alternatively they could be executed locally by the master node(s). The latter case does not imply that execution is sequential, because a master node can make use of some other parallel runtime (e.g., MPI).

On a Spark cluster, the main steps are executed within two Spark stages that run on a set of worker nodes. A *stage* is a set of independent tasks executing functions that do not need to perform data shuffling (e.g., transformation and action functions). Specifically: data filtering and mapping are executed within the first stage (*Stage 0*), while data partitioning and reduction are executed within the second stage (*Stage 1*). Concerning the remaining steps (data acquisition,

<sup>3</sup> <https://mahout.apache.org/>

<sup>4</sup> <https://spark.apache.org/mlib/>

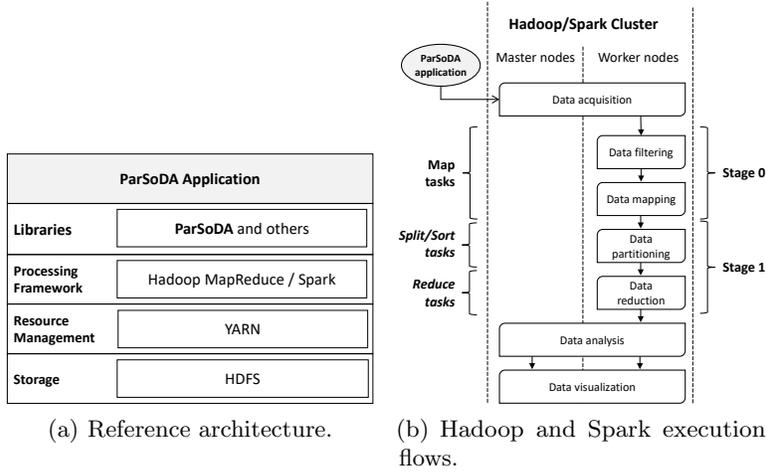


Fig. 4: Reference architecture and execution flow.

data analysis, and data visualization), the same considerations made for Hadoop apply to Spark.

### 3.2 Usability and scalability evaluation

Writing a parallel data analysis application from scratch usually requires deep programming skills and the writing of many lines of code. In fact, designing and implementing such kind of applications pose a number of challenges to developers such as parallelization of complex algorithms, reduction of communication costs, and optimization of memory usage. As demonstrated in [13], using ParSoDA leads to a drastic reduction of lines of code. In particular, ParSoDA allows programmers to save hundred lines of code in the main (as the programmer needs to specify only the functions to be used and their parameters), in the data acquisition and data partition steps (where built-in functionalities are exploited), as well as in the data filtering, mapping, and reduction steps (where the programmer needs only to define the function logic). For the data analysis and visualization steps, we used the same code to invoke external libraries, which does not lead to a gain in terms of lines of code. However, for these steps, ParSoDA ensures many advantages in terms of usability. In fact, in the application main defined through ParSoDA, all the MapReduce jobs created for the different steps, such as the ones in the analysis and visualization steps, are automatically chained. This means that the output of a job is automatically used as input to the next step. In contrast, without ParSoDA, programmers need to manually control the execution flow among different jobs.

The scalability of ParSoDA has been evaluated by running the data analysis applications on a private cloud infrastructure with 300 cores and 1.2 TB of

RAM. In the experiments, the Spark version of ParSoDA has been used, since, as demonstrated in [10], it resulted to be faster than the Hadoop version of the library.

## 4 Nubytics

Nubytics [16] is a system that exploits Cloud facilities to provide scalable services for analyzing large datasets. The system allows users to import their data to the Cloud, extract knowledge models using high performance data mining services, and use the inferred knowledge to predict new data. Nubytics provides data classification and regression services that can be used in a variety of scientific and business applications. Scalability is ensured by a parallel computing approach that fully exploits the resources available on the Cloud. In addition, Nubytics is provided in accordance with the Software-as-a-Service (SaaS) model. This means that no installation is required on the user’s machine: the Nubytics interface is offered by a web browser, so it can be run from most devices, including desktop PCs, laptops, and tablets. This is a key feature for users who need ubiquitous and seamless access to scalable data analysis services, without needing to cope with the installation and system management issues of traditional analytics tools.

Nubytics differs from general purpose data analysis frameworks like Azure ML, Hadoop and Sparks, or data-oriented workflow management systems like CloudFlows and DMCF, as it provides specialized services for data classification and prediction. These services are provided by a Web interface that allows data analysts to focus on the data analysis process without worrying on low level programming details. This approach is similar to that adopted by BigML. However, Nubytics also focuses on scalability, by implementing an ad hoc parallel computing approach that fully exploits the distributed resources of a Cloud computing platform.

### 4.1 Architecture

The Nubytics architecture includes storage and compute components. The storage components are:

- *Data Folder* that contains data sources and the results of data analysis, and *Tool Folder* that contains algorithms for data analysis and prediction.
- *Data Table*, *Tool Table* and *Task Table* that contain metadata information associated with data, tools, and tasks.
- *Task Queue* that contains the tasks to be executed.

The compute components are:

- *Virtual Compute Servers* that execute the data analysis tasks.
- *Virtual Web Servers* that host the system front end, i.e., the Nubytics web interface.

The architecture manages submission and execution of data analysis tasks by the following steps:

1. Using the services provided by the front end, the user can configure and submit the desired data analysis task (e.g., training a classification model from a dataset).
2. Exploiting a data parallel approach, the system models the task as a set of parallel sub-tasks that are inserted into the Task Queue for processing.
3. Each idle Virtual Compute Server picks a sub-task from the Task Queue and concurrently starts its execution.
4. Each Virtual Compute Server gets the part of data assigned to it from the Data Folder where the original dataset is stored.
5. After sub-task completion, each Virtual Compute Server puts the result on the Data Folder.
6. The front end notifies the user as soon as the task has completed, and allows her/him to access the results.

## 4.2 Services

The Nubytics front end is divided into three sections - *Datasets*, *Tasks* and *Models* - corresponding to the three groups of services provided by the system: dataset management, task management and model management.

The datasets of a user are stored in a Cloud storage space associated to the user's account. The *Datasets* section provides several data management services, including: importing (uploading) a dataset from the user's terminal; exporting (downloading) a dataset to the user's terminal; listing and searching the available datasets; modifying the metadata of a dataset; creating a copy, deleting, or restoring a dataset.

The *Tasks* section provide services for configuring, submitting and managing data analysis tasks. Two classes of tasks can be submitted: training tasks and prediction tasks.

A training task takes as input a dataset and produces a classification or regression model from it. The goal of classification is to derive a model that classifies the instances of a dataset into one or more classes. Using a classification model, we can predict the membership of a new data instance to a given class from a set of predefined classes. The goal of regression is to build a model that associates a numerical value to the instances of a dataset. Therefore, a regression model can be used to forecast a quantitative value starting from the field values of a new data instance.

The configuration of a training task is made by selecting the input dataset, the class field (which is categorical in case of classification and numerical in case of regression), and the predictive fields that must be considered for the analysis (they can be all - or a subset of - the original dataset fields). A parallel computing approach is used to speedup the execution of training tasks. This is done using a data parallel approach that divides the original task in sub-tasks, assigns a

sub-task to a different virtual compute server on the Cloud, and joins the partial results computed by multiple servers into a single model.

A prediction task takes two input elements: a model generated by a training task, and a new dataset whose instances must be classified or regressed. As a result, the new dataset will include a new field containing the predicted class label (in case of classification) or numerical value (in case of regression) of each tuple. Also in this case, parallelism is exploited by performing the prediction task in parallel on multiple Cloud servers.

Multiple tasks can be submitted to the system, and the user can monitor the status of each one through a task management interface, as shown by the screenshot in Figure 5. For each task, the interface shows the task type (prediction or training), some information about execution (start, end, and elapsed time), and the current status. Additional details on a task can be seen by selecting the corresponding row. For instance, the figure shows Input Dataset and Output Model of the second task, which is a Training task.

#	Type	Started	Ended	Elapsed	Status
1	Prediction	03/24/2016 12:56:53		00:00:13	Running
2	Training	03/24/2016 12:54:35		00:02:30	Running
		Input Dataset: Census_Income		Output Model: Census_Income_Model	
3	Training	03/23/2016 10:00:45	03/23/2016 10:38:39	00:37:53	Done
4	Training	03/23/2016 08:40:12	03/23/2016 09:55:50	01:15:38	Done
5	Training	03/22/2016 18:17:00	03/22/2016 18:21:28	00:04:27	Stopped
6	Creation	03/22/2016 17:57:41	03/22/2016 18:02:50	00:05:08	Done
7	Training	03/22/2016 13:04:56	03/22/2016 13:09:21	00:04:24	Stopped
8	Prediction	03/21/2016 15:20:50	03/21/2016 15:21:13	00:00:23	Done
9	Creation	03/21/2016 15:08:51	03/21/2016 15:08:57	00:00:06	Done
10	Modification	03/21/2016 12:30:13	03/21/2016 12:30:32	00:00:19	Done

Fig. 5: Screenshot of the Tasks section.

## 5 SMA4TD

SMA4TD (Social Media Analysis for Trajectory Discovery) [17] is a methodology aimed at discovering behavior rules, correlations and mobility patterns of visitors attending large-scale events, through the analysis of a large number of social media posts. In particular, the main goals of the methodology are as follows.

1. *Discovery of most visited places and most attended events.* We analyze the collected data to discover the places that have been most visited by users, and the events that have been most attended by visitors during the observed period.

2. *Discovery of most frequent sets of visited places and most frequent sets of attended events.* We extract the sets of places that are most frequently visited together by users, and the events that have been most attended by visitors during the observed period.
3. *Discovery of most frequent mobility patterns among places and most frequent sequences of attended events.* We analyze the collected data to discover mobility behaviors among the places, and to extract useful knowledge (i.e. patterns, rules and regularities) about the attended events.
4. *Discovery of the origin and destination of visitors.* We study the mobility flows of people attending the events, evaluating which countries visitors came from and which countries they moved after the events. In some cases, this information can give some insights about the touristic impact on the local territory.

The methodology is composed of seven steps: *i*) identification of the set of events; *ii*) identification of places-of-interests where the events take place; *iii*) collection of geotagged items related to events and pre-processing; *iv*) identification of users who published at least one of the geotagged items; *v*) pre-processing and creation of the input dataset; *vi*) data analysis and trajectory mining; and *vii*) results visualization.

### 5.1 Steps 1-2: Definition of events and places-of-interest

The first two steps aim at defining the events  $\mathcal{E}$  and the corresponding places-of-interest  $\mathcal{P}$ . Specifically, during step 1, each event is described by the id of the place-of-interest (PoI) where it is located, starting/ending time of the event, and other optional data (e.g., free/paid event, type of event, etc.). Step 2 is aimed at defining the geographical boundaries of the PoIs in  $\mathcal{P}$ . This can be done in two ways: *i*) *manually* defining the boundaries of the PoIs (e.g., as polygons on a map); *ii*) *automatically*, using external services (e.g., cadastral maps [19]), or public web services providing the geographical boundaries of a place given its name (e.g., OpenStreetMap<sup>5</sup>).

### 5.2 Steps 3-4-5: Collection and pre-processing of geotagged items, identification of users and creation of the input dataset

The goal of step 3 is to collect all the geotagged items  $\mathcal{G}$  posted during each event  $e_i \in \mathcal{E}$  from the place  $p_i$  where  $e_i$  was held. Data collection is done by using the publicly available APIs provided by most social media platforms. The  $\mathcal{G}$  dataset is pre-processed in order to clean, select and transform data to make it suitable for analysis. In particular, we first clean the collected data by removing all items with unreliable positions (e.g., items with coordinates that have been manually set by users or applications). Then, we proceed by selecting only the geotagged items posted by users who actually attended an event, by removing replies and

<sup>5</sup> <https://www.openstreetmap.org/>

favorites posted by other users. Finally, we transform data by keeping one item per user per event, because we are interested to know only if a user attended an event or not. The identification of users is the goal of step 4. This is done by extracting the set  $\mathcal{U}$  of distinct users who published at least one geotagged item in  $\mathcal{G}$ .

Step 5 creates the input datasets  $\mathcal{D} = \{d_1, d_2, \dots\}$ , where  $d_i$  is a tuple  $\langle u_i, \{e_{i1}, e_{i2}, \dots, e_{ik}\}, optFields \rangle$  in which  $e_{ij}$  is the  $j^{th}$  event attended by user  $u_i$ , and *optFields* are optional descriptive fields (e.g., nationality, interests).

### 5.3 Step 6-7: Data mining and results visualization

After having built the input dataset  $\mathcal{D}$ , it is analyzed for discovering behaviour and mobility patterns of users attending the large-scale event under investigation. Specifically, we perform both *associative* and *sequential analysis*, as described in the following.

*Associative analysis* is exploited with the goal of discovering (inside data) the item values that occur together with a high frequency. The mechanisms of association allow identifying the conditions that tend to occur simultaneously, or the patterns that repeat in certain conditions. Applied to dataset  $\mathcal{D}$ , we perform two associative mobility mining tasks: (i) *frequent event sets discovery*, aimed at extracting the sets of events (places) that are most frequently attended (visited) together by visitors during the whole observed large-scale event; and (ii) *frequent event rules extraction*, devoted to discover frequent associative rules among the events.

On the other hand, *sequential analysis* algorithms are intended to discover the sequences of elements that occur most frequently in the data. Unlike associative analysis, in sequential analysis are fundamental the time dimension and the chronological order in which the values appear in the data. In our case, this type of analysis is useful to discover the most frequent mobility patterns among the places, and/or the most frequent sequences of attended events. Moreover, if the observed period is extended to some days (or weeks) before/after the event time, we can also discover the origin/destination (i.e., country, city) of visitors and which countries visitors came from/move after the event (i.e., to infer touristic insights).

Finally, results visualization is performed by the creation of info-graphics aimed at presenting the results in a way that is easy to understand to the general public, without providing complex statistical details that may be hard to understand to the intended audience. The graphic project is grounded on some of the most acknowledged and ever-working principles underpinning a 'good' info-graphic piece.

### 5.4 Study cases: FIFA World Cup 2014 and EXPO 2015

In this section we present the results obtained by analyzing geotagged posts of social media users attending the FIFA World Cup 2014 and EXPO 2015.

**FIFA World Cup 2014** During the FIFA World Cup 2014, we monitored the Twitter users attending the 64 matches played during the football competition and analyzed such data through the SMA4TD methodology to discover behaviors and frequent movements of fans [14]. In this case study, the places-of-interest are the stadiums in which the World Cup matches have been played. The corresponding RoIs have been manually defined from a map as the smallest rectangles fully containing the boundaries of each stadium. For each match, we considered only the tweets posted from coordinates falling within the above defined RoIs during the matches. Totally, the number of tweets that have been collected (from June 12 to July 13, 2014) amounted to about 526,000. We have made several analyzes on user behavior. For example, we described how the number of people attending the matches changed over time. To do that, we report in Figure 6 trends and numbers *(i)* of Twitter users we tracked attending at the matches during the World Cup, and *(ii)* of attendees officially published by the FIFA website<sup>6</sup>. Specifically, Figure 6 shows a time plot of the collected attendance data, in which the number of attendees is plotted versus the number of matches.

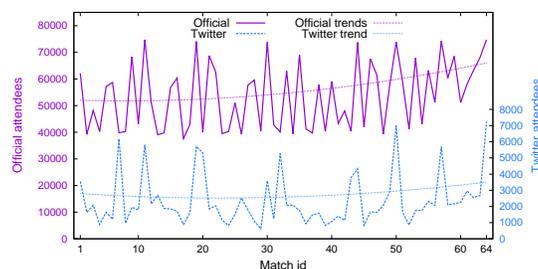


Fig. 6: Number of attendees per match, comparing Twitter users and official attendee numbers.

It clearly shows that there are several peaks of participation during the competition, probably corresponding to some matches that have attracted more attention with respect to other ones. Interestingly, in some cases Twitter data peaks are equivalent to the official attendance ones. We also studied the participation of fans to the matches. The results show that 71.3% of the fans attended a single match, 16% attended two matches, 6% attended three matches, and only 6.7% attended four or more matches. We also studied the most frequent paths of fans who attended two or three matches of the same team during the group stage. For example, the most frequent 2-match-set was  $\langle \text{Colombia-Greece}, \text{Colombia-Cote d'Ivoire} \rangle$ , followed by  $\langle \text{Brazil-Mexico}, \text{Croatia-Mexico} \rangle$ , and by  $\langle \text{Argentina-Bosnia}, \text{Argentina-Iran} \rangle$ , i.e., matches likely attended by fans of Colombia, Mexico and Argentina. Looking at their nationality, spectators were likely fans of Mexico, Brazil and Australia.

<sup>6</sup> <http://www.fifa.com/worldcup/archive/brazil2014>

**EXPO 2015** For the second study case, the EXPO 2015, we monitored Instagram users who visited EXPO pavilions to discover mobility patterns inside the exhibition area, correlations among visits to pavilions and the main flows of origin/destination of visitors [15]. EXPO 2015<sup>7</sup> was a Universal Exposition held under the theme “Feeding the Planet, Energy for Life”, which was hosted in Milan, Italy, from May 1<sup>st</sup> to October 31<sup>st</sup>, 2015. Exhibitors were individual countries, international organizations, civil society organizations and companies, for a total of 188 exhibition spaces. Some of the exhibitors were hosted inside individual (self-built) pavilions, while others were hosted inside shared pavilions. For the sake of uniformity, in this paper we will use the term pavilion to indicate both an individual pavilion and a distinct area (assigned to a given exhibitor) of a shared pavilion. Cumulatively, about 22.2 million people visited the EXPO area and its pavilions during the six months of the whole exposition, making it the world-wide largest event of the year 2015. Visitors at EXPO used various social network to share their experience with friends and followers.

The set of events  $\mathcal{E}$  considered for this scenario is composed by the showcases (each one organized by a country or organization/company) exhibited in the exposition spaces (generally referred as pavilions in the following). Specifically, let us consider  $\mathcal{E} = \{e_1, e_2, \dots, e_{188}\}$ , where each  $e_i$  is described by the following properties:

$$e_i = \langle p_i, [t_i^{begin}, t_i^{end}] \rangle$$

where  $p_i$  is the pavilion,  $t_i^{begin}$  is May 1st and  $t_i^{end}$  is October 31st.

The places-of-interest to be considered are the pavilions. Specifically, we defined the PoI set  $\mathcal{P} = \{p_1, p_2, \dots, p_{188}\}$ , where each  $p_i$  is a pavilion that has been used as exhibition area during the EXPO 2015. For each PoI, we drew its corresponding RoI as a rectangle bounding the pavilion area.

Figure 7 shows a comparison between trends and numbers of the Instagram visitors we tracked, and the official visitors published on the EXPO website<sup>8</sup>. The observed period is August 1<sup>st</sup> - October 31<sup>st</sup>, but official numbers have been published only for the period starting in August, thus the corresponding curve has been traced only for the last three months. We used different scales for Instagram visitor numbers and the EXPO visitor ones: on the right is the scale of the formers, while on the left is the scale of the latter ones. In particular, Figure 7(a) shows a time plot of the daily visits to EXPO. The trends are quite evident: initially (May and June) the visitors are relatively few; then, they grow significantly during the months of September and October. Moreover, there are several peaks of attendance, corresponding to visits occurred during the week-end days. By looking at the trends in the figure, it can be noted a strong correlation (Pearson coefficient 0.7) between official visitor numbers and those obtained from our analysis, which confirms the reliability of the results we obtained. Figure 7(b) compares Instagram and official visitor numbers, aggregated by the week day (Pearson correlation 0.94). The results clearly show that during the week-end

<sup>7</sup> <http://www.expo2015.org/>

<sup>8</sup> <http://www.expo2015.org/>

days there is a peak of visits, with the highest number of people registered on Saturdays.

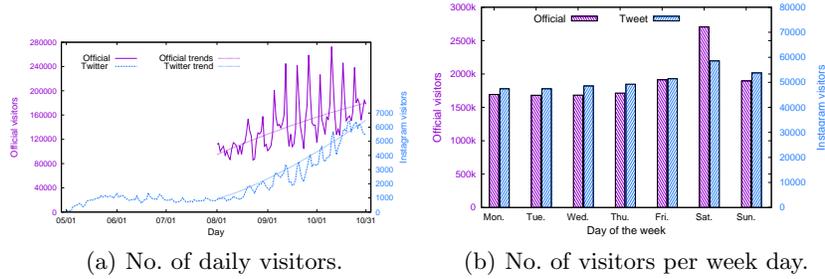


Fig. 7: Statistics about visitors, comparing Instagram users and official attendee numbers.

## 6 RoI mining

Geotagged data gathered from social media can be used to discover interesting locations visited by users called Places-of-Interest (PoIs). Since a PoI is generally identified by the geographical coordinates of a single point, it is hard to match it with user trajectories. Therefore, it is useful to define an area, called *Region-of-Interest (RoI)*, to represent the boundaries of the PoI's area. *RoI mining* techniques are aimed at discovering Regions-of-Interest from PoIs and other data.

*G-RoI* [11] is a novel RoI mining technique that exploits the indications contained in geotagged social media items (e.g. tweets, posts, photos or videos with geospatial information) to discover the RoI of a PoI with a high accuracy. Given a PoI  $p$  identified by a set of keywords, a geotagged item is associated to  $p$  if its text or tags contain at least one of those keywords. Starting from the coordinates of all the geotagged items associated to  $p$ , *G-RoI* calculates an initial convex polygon enclosing all such coordinates, and then iteratively reduces the area using a density-based criterion. Then, from all the convex polygons obtained at each reduction step, *G-RoI* adopts an area-variation criterion to choose the polygon representing the RoI for  $p$ .

Let a PoI  $\mathcal{P}$  be identified by one or more keywords  $K = \{k_1, k_2, \dots\}$ . Let  $G_{all}$  be a set of geotagged items. Let  $G = \{g_0, g_1, \dots\}$  be the subset of  $G_{all}$ , obtained by applying a *G-RoI preprocessing* procedure that selects from  $G_{all}$  only the geotagged items associated to  $\mathcal{P}$ , i.e., the text or tags of each  $g_i \in G$  contains at least one keyword in  $K$ . Let  $C = \{c_0, c_1, \dots\}$  be a set of coordinates, where  $c_i$  represents the coordinates of  $g_i \in G$ . Thus, every  $c_i \in C$  represents the coordinates of a location from which a user has created a geotagged item referring to  $\mathcal{P}$ . Let  $cp_0$  be a convex polygon enclosing all the coordinates in  $C$ ,

obtained by running the convex hull algorithm [3] on  $C$ , described by a set of vertices  $\{v_0, v_1, \dots\}$ .

To find the RoI  $\mathcal{R}$  for  $\mathcal{P}$ , the G-RoI algorithm uses two main procedures:

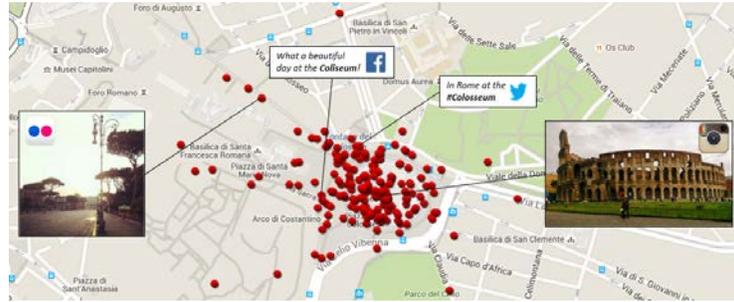
- *G-RoI reduction*. Starting from  $cp_0$ , it iteratively reduces the area of the current convex polygon by deleting one of its vertex. A density-based criterion is adopted to choose the next vertex to be deleted. The density of a polygon is the ratio between the number of geotagged items enclosed by the polygon, and its area. At each step, the procedure deletes the vertex that produces the polygon with highest density, among all the possible polygons. The procedure ends when it cannot further reduce the current polygon, and returns the set of convex polygons  $CP = \{cp_0, \dots, cp_n\}$  obtained after the  $n$  steps that have been performed.
- *G-RoI selection*. It analyses the set of convex polygons  $CP$  returned by the *G-RoI reduction* procedure, and selects the polygon representing RoI  $\mathcal{R}$  for PoI  $\mathcal{P}$ . An area-variation criterion is adopted to choose  $\mathcal{R}$  from  $CP$ . Given  $CP$ , the procedure identifies two subsets: a first subset  $\{cp_0, \dots, cp_{cut-1}\}$  such that the area of any  $cp_i$  is significantly larger than the area of  $cp_{i+1}$ ; a second subset  $\{cp_{cut}, \dots, cp_n\}$  such that the area of any  $cp_i$  is not significantly larger than the area of  $cp_{i+1}$ . The procedure returns  $cp_{cut}$  as RoI  $\mathcal{R}$ . This corresponds to choosing  $cp_{cut}$  as the corner point of a discrete *L-curve* [20] obtained by plotting the areas of all the convex polygons in  $CP$  on a Cartesian plane, as detailed later in this section.

## 6.1 Methodology

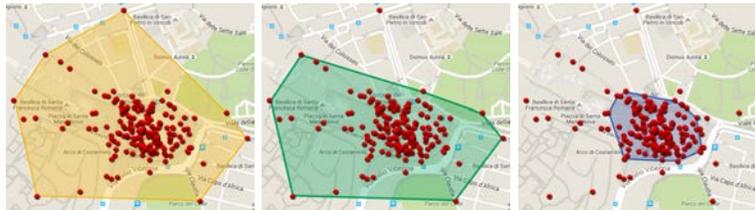
Without going into algorithmic details, which can be found at [11], we briefly describe how the *G-RoI reduction* and *selection* procedures work through a real example. Starting from a small sample of 200 geotagged items from different social media, referring to the *Colosseum* in Rome and posted at a maximum distance of 500m from it.

In their posts and photos, social media users identify the *Colosseum* with different keywords, such as *Coliseum*, *Coliseo*, *Colisée*, and synonymous such as *Flavian Amphitheatre* or *Amphitheatrum Flavium*. All the geotagged items in our sample contain at least one of such keywords. From these posts, the 200 coordinates shown in Figure 8(a) have been extracted. Given the coordinates, the *G-RoI reduction* procedure calculates the initial convex polygon  $cp_0$  (shown Figure 8(b)), and then iteratively reduces the area. Figure 8(c) shows polygon  $cp_1$  obtained after the first step by deleting one of the vertices from  $cp_0$ . The *G-RoI reduction* procedure iterates until it cannot further reduce the current polygon. The output of the procedure is the set of convex polygons  $CP = \{cp_0, cp_1, \dots, cp_n\}$  obtained at each step.

The *G-RoI selection* procedure identifies the point  $p_{cut}$  that is located at the maximum distance ( $dist^{max}$ ) from the *reference line* joining the first point and the last point under analysis ( $p_0$  and  $p_n$ ). If the set of points  $\{p_{cut}, \dots, p_n\}$  follows a linear trend, i.e., there is no point below a *threshold line* at distance  $th$  from



(a) Collection of geotagged items.



(b) Initial convex polygon  $cp_0$ . (c) Generating  $cp_1$  by deleting one vertex from  $cp_0$ . (d) G-RoI selection of Colosseum's convex polygons.

Fig. 8: G-RoI reduction and selection on Colosseum's geotagged items.

the reference line joining the points  $p_{cut}$  and  $p_n$ , then the procedure returns the polygon corresponding to  $p_{cut}$  as RoI  $\mathcal{R}$  (see Figure 8(d)). Otherwise, the *G-RoI selection* procedure iterates by finding a new cut-off point from the set of points on the right of  $p_{cut}$ .

We experimentally evaluated the accuracy of G-RoI in detecting the RoIs associated to a set of PoIs. The analysis was carried out on 24 PoIs located in the center of Rome (St. Peter's Basilica, Colosseum, Circus Maximus, etc.) using about 1.2 millions geotagged items published in Flickr from January 2006 to May 2016 in the areas under analysis. Specifically, we made several preliminary tests to find parameter values that perform effectively in that scenario, taking into account that the various PoIs are characterized by significant variability of shape, area and density (number of Flickr photos divided by area). In particular, the threshold  $th$  was set to 0.27. The experimental results showed also that G-RoI is able to detect RoIs with high accuracy. Over a set of 24 PoIs in Rome, G-RoI achieved a mean precision of 0.78, a mean recall of 0.82, and a mean  $F_1$  score of 0.77.

## 7 Iterative Opinion Mining using Neural Networks

*IOM-NN (Iterative Opinion Mining using Neural Networks)* [5] is a new methodology for estimating the polarization of public opinion on political events characterized by the competition of factions or parties. It can be considered as an alternative technique to traditional opinion polls, since it is able to capture the opinion of a larger number of people more quickly and at a lower cost. In particular, IOM-NN uses an automatic incremental procedure based on feed-forward neural networks for analyzing the posts published by social media users. Starting from a limited set of classification rules, created from a small subset of hashtags that are notoriously in favor of specific factions, our methodology iteratively generates new classification rules. A classification rule allows to determine if a post is in favor of a faction based on the words/hashtags it contains. Then, such rules are used to determine the polarization of social media users - who wrote posts about the political event - towards a faction. As shown in Figure 9, the proposed methodology consists of three main steps:

1. *Collection of posts*: posts are collected by using a set of keywords related to the selected political event.
2. *Classification of posts*: the collected posts are then classified by using an incremental procedure implemented through neural networks.
3. *Polarization of users*: the classified posts are analyzed for determining the polarization of users towards a faction.

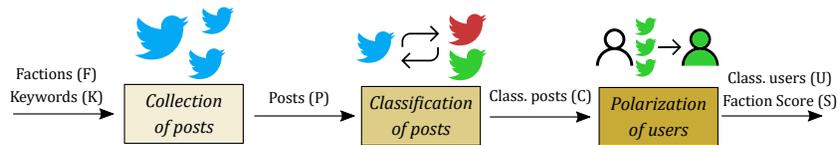


Fig. 9: Execution flow of IOM-NN.

**Collection of posts** A political event  $\mathcal{E}$  is characterized by the rivalry of different factions  $F = \{f_1, f_2, \dots, f_n\}$ . Examples of political events and relative factions are: *i*) municipal election, in which a faction supports a mayor candidate; *ii*) parliament election, in which a faction supports a party; *iii*) presidential election, in which a faction supports a presidential candidate [21]. The posts are collected by using the keywords that people commonly use to refer the political event  $\mathcal{E}$  on social media. Such keywords  $K$  can be divided in two groups:

- $K_{context}$ , which contains generic keywords that can be associated to  $\mathcal{E}$  without referring to any specific faction in  $F$ .

- $K_F^\oplus = K_{f_1}^\oplus \cup \dots \cup K_{f_n}^\oplus$ , where  $K_{f_i}^\oplus$  contains the keywords used for supporting  $f_i \in F$  (positive faction keywords).

The keywords in  $K$  are given as input to public APIs provided by social media platforms, which permit to *collect* posts containing one or more keywords.

The collected posts are *pre-processed* before the analysis. The output of this step is a collection of posts  $P$ . In particular, they are modified and filtered as follows:

- The text of posts is normalized by transforming it to lowercase and replacing accented characters with regular ones (e.g., IOVOTOSI or iovotosí  $\rightarrow$  iovotosi).
- Words are stemmed for allowing matches with declined forms (e.g., vote or votes or voted  $\rightarrow$  vot).
- Stop words are removed from text by using preset lists.
- All the posts written in a language different from the one(s) spoken in the nation(s) hosting the considered political event are filtered out.

**Classification of posts** The input of the algorithm for the classification of posts is composed of: the posts  $P$  generated in the previous step, the set of positive faction keywords  $K_F^\oplus$ , the maximum number of iterations  $max\_iters$ , the minimum increment of the classified posts  $eps$  at each iteration, and a threshold  $th$ . Instead, the output is a collection of posts  $C$  that have been classified in favor of a faction.

As discussed in [4], the algorithm is divided in two parts. The first part performs the preliminary iteration (iteration 0). At this iteration, IOM-NN exploits the set of positive faction keywords ( $K_F^\oplus$ ) for classifying a part of the posts. Specifically, it classifies a post in favor of a faction if it contains only positive keywords for such faction. In general, at the end of this iteration, only a small part of posts are classified, since not all users use keywords in  $K_F^\oplus$  for declaring their support to factions. The second part iteratively generates new classification rules for classifying other posts. At each iteration, such rules are inferred by exploiting the posts that have been classified at the previous iterations.

**Polarization of users** This algorithm is used for determining the polarization of users. The input is composed of: a collection of classified posts  $C$ , a filtering function  $filter$  with its parameters  $par_f$ , and a polarization function  $polarize$  with its parameters  $par_p$ . The output is composed of a collection of classified users  $U$  and a faction score ( $S$ ) containing the polarization percentages for each faction. As first step, the classified posts are aggregated by user to produce a dictionary ( $C_U$ ), which contains the list of classified posts  $P_u$  for each user  $u$ . Two empty variables are initialized for storing the output. On each pair  $\langle u, P_u \rangle$  of  $C_U$ , the algorithm performs the following operations:

- It filters out all the pairs that do not match the criteria defined by the  $filter$  function. For example, users who published a number of posts below a given threshold are skipped.

- Using the classified posts  $P_u$ , it computes  $v_s^u$  a vector containing the score of user  $u$  for each faction. The score vector is calculated by using the function *polarize*.
- It adds the pair  $\langle u, v_s \rangle$  to  $U$ .

Then, the algorithm calculates the overall faction score  $S$  as the normalized sum of the user vector scores  $\langle u, v_s^u \rangle$ . Finally, the output is returned.

### 7.1 Case study: the 2016 US presidential election

In this section we describe and analyze a case study: the 2016 US presidential election, which was characterized by the rivalry between Hillary Clinton and Donald Trump. The analysis has been performed on data collected for ten US Swing States: Colorado, Florida, Iowa, Michigan, Ohio, New Hampshire, North Carolina, Pennsylvania, Virginia, and Wisconsin. Overall about 2.5 million of tweets, posted by 521,291 users, have been collected from October 10, 2016 to November 7, 2016 (the day before the election). From such data we filtered out all the tweets posted by users with a not defined location or with a location that does not belong to any of the considered states. In particular, for each faction  $f_i$  we defined three set of keywords  $K_{f_i}^{\oplus}$ ,  $K_{f_i}^{\ominus}$  and  $K_{f_i}^{\circ}$  that are respectively positive, negative and neutral keywords for faction  $f_i$ . For example, for the *Hillary Clinton* faction  $K_{Clinton}^{\oplus}$  contains keywords used to clearly support her party (e.g., *#voteHillary*),  $K_{Clinton}^{\ominus}$  contains keywords to speak negatively about her (e.g., *#neverhillary*),  $K_{Clinton}^{\circ}$  contains neutral keywords (e.g., *clinton* or *democrats*). *IOM-NN* exploits only positive faction keywords ( $K_{f_i}^{\oplus}$ ) for classifying posts and then for determining the polarization of users.

For such study case, the *filter* and *polarize* functions have been configured as follows. Specifically, a user  $u$  is considered only if he/she fulfills the following criteria: *i*)  $u$  posted at least *minPosts* on the political event of interest; *ii*) it exists a faction  $f$  for which  $u$  has published more than 2/3 of his/her posts. For each user  $u$ , the *polarize* function returns a vector score as follows: the percentage of posts written by  $u$  in favor of preferred faction  $f$ , 0 for the other factions.

Figure 10 shows how the user polarization algorithm works on some classified posts. For each user, the posts if favor of Clinton and Trump are counted. Users who fulfill the criteria of filter function are considered and added to the set of

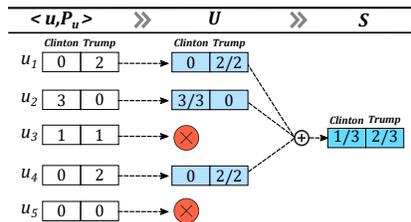


Fig. 10: Example of how the *user polarization* algorithm works.

classified users  $U$ . Then  $U$  is combined and normalized to obtain the vector  $S$  containing the overall polarization percentages.

As shown in Figure 11, IOM-NN is able to correctly identify the winning candidate in 8 out of 10 cases, outperforming the opinion polls that correctly classifies 6 out 10 states.

	Colorado	Florida	Iowa	Michigan	New Hampshire	North Carolina	Ohio	Pennsylvania	Virginia	Wisconsin
Real										
IOM-NN										
Opinion polls										

Fig. 11: Comparison among the real winning candidate and that identified by IOM-NN and opinions polls. The *Democratic Donkey* symbolizes the party of Hillary Clinton, while the *Republican Elephant* that of Donald Trump.

## 8 Conclusions

In science and business, scientists and professionals analyze huge amounts of data, commonly called Big Data, to extract information and knowledge useful for making new discoveries or for supporting decision processes. This can be done by exploiting Big Data analytics techniques and tools. In this scenario, Cloud computing represents a compelling solution for Big Data analytics, allowing faster data analysis, that means more timely results and then greater data value. This paper presented some recent Cloud-based frameworks and methodologies for Big Data processing that can be used for developing and executing different kind of data analysis applications. In particular, in the first part of the paper we presented some tools for developing and running Big Data application on Cloud (i.e., DMCF, ParSoDA, and Nubytics). Instead, in the second part we present some innovative methodologies for extracting useful information about mobility behaviors (i.e., SMA4TD and G-RoI) and political sentiment of social media users (i.e., IOM-NN).

## References

1. Agapito, G., Cannataro, M., Guzzi, P., Marozzo, F., Talia, D., Trunfio, P.: Cloud4snp: Distributed analysis of snp microarray data on the cloud. pp. 468–475 (2013). <https://doi.org/10.1145/2506583.2506605>
2. Altomare, A., Cesario, E., Comito, C., Marozzo, F., Talia, D.: Trajectory pattern mining for urban computing in the cloud. *IEEE Transactions on Parallel and Distributed Systems* **28**(2), 586–599 (2017). <https://doi.org/10.1109/TPDS.2016.2565480>

3. Barber, C.B., Dobkin, D.P., Huhdanpaa, H.: The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.* **22**(4), 469–483 (Dec 1996)
4. Belcastro, L., Cantini, R., Marozzo, F., Talia, D., Trunfio, P.: Discovering political polarization on social media: A case study. pp. 182–189 (2019). <https://doi.org/10.1109/SKG49510.2019.00038>
5. Belcastro, L., Cantini, R., Marozzo, F., Talia, D., Trunfio, P.: Learning political polarization on social media using neural networks. *IEEE Access* **8**, 47177–47187 (2020). <https://doi.org/10.1109/ACCESS.2020.2978950>
6. Belcastro, L., Marozzo, F., Talia, D.: Programming models and systems for big data analysis. *International Journal of Parallel, Emergent and Distributed Systems* **34**(6), 632–652 (2019). <https://doi.org/10.1080/17445760.2017.1422501>
7. Belcastro, L., Marozzo, F., Talia, D., Trunfio, P.: Programming visual and script-based big data analytics workflows on clouds. *Advances in Parallel Computing* **26**, 18–31 (2015). <https://doi.org/10.3233/978-1-61499-583-8-18>
8. Belcastro, L., Marozzo, F., Talia, D., Trunfio, P.: Using scalable data mining for predicting flight delays. *ACM Transactions on Intelligent Systems and Technology* **8**(1) (2016). <https://doi.org/10.1145/2888402>
9. Belcastro, L., Marozzo, F., Talia, D., Trunfio, P.: Big data analysis on clouds (2017). [https://doi.org/10.1007/978-3-319-49340-4\\_4](https://doi.org/10.1007/978-3-319-49340-4_4)
10. Belcastro, L., Marozzo, F., Talia, D., Trunfio, P.: Appraising spark on large-scale social media analysis. *Lecture Notes in Computer Science* **10659 LNCS**, 483–495 (2018). [https://doi.org/10.1007/978-3-319-75178-8\\_39](https://doi.org/10.1007/978-3-319-75178-8_39)
11. Belcastro, L., Marozzo, F., Talia, D., Trunfio, P.: G-roi: Automatic region-of-interest detection driven by geotagged social media data. *ACM Transactions on Knowledge Discovery from Data* **12**(3) (2018). <https://doi.org/10.1145/3154411>
12. Belcastro, L., Marozzo, F., Talia, D., Trunfio, P.: Parsoda: high-level parallel programming for social data mining. *Social Network Analysis and Mining* **9**(1) (2019). <https://doi.org/10.1007/s13278-018-0547-5>
13. Belcastro, L., Marozzo, F., Talia, D., Trunfio, P.: A high-level programming library for mining social media. In: *Post-Proc. of the High Performance Computing Workshop 2018, Cetraro, Italy, Advances in Parallel Computing. Advances in Parallel Computing*, vol. 34, pp. 3–21. IOS Press (2-6 July 2019)
14. Cesario, E., Congedo, C., Marozzo, F., Riotta, G., Spada, A., Talia, D., Trunfio, P., Turri, C.: Following soccer fans from geotagged tweets at fifa world cup 2014. pp. 33–38 (2015). <https://doi.org/10.1109/ICSDM.2015.7298021>
15. Cesario, E., Iannazzo, A., Marozzo, F., Morello, F., Riotta, G., Spada, A., Talia, D., Trunfio, P.: Analyzing social media data to discover mobility patterns at expo 2015: Methodology and results. pp. 230–237 (2016). <https://doi.org/10.1109/HPCSim.2016.7568340>
16. Cesario, E., Iannazzo, A., Marozzo, F., Morello, F., Talia, D., Trunfio, P.: Nubytics: Scalable cloud services for data analysis and prediction (2016). <https://doi.org/10.1109/RTSI.2016.7740643>
17. Cesario, E., Marozzo, F., Talia, D., Trunfio, P.: Sma4td: A social media analysis methodology for trajectory discovery in large-scale events. *Online Social Networks and Media* **3-4**, 49–62 (2017). <https://doi.org/10.1016/j.osnem.2017.10.002>
18. Duro, F.R., Blas, J.G., Carretero, J.: A hierarchical parallel storage system based on distributed memory for large scale systems. In: *Proceedings of the 20th European MPI Users' Group Meeting*. pp. 139–140 (2013)
19. de Graaff, V., de By, R.A., van Keulen, M., Flokstra, J.: Point of interest to region of interest conversion. In: *Proceedings of the 21st ACM SIGSPATIAL*

- International Conference on Advances in Geographic Information Systems. pp. 388–391. SIGSPATIAL'13, ACM, New York, NY, USA (2013)
20. Hansen, P.C.: Analysis of discrete ill-posed problems by means of the L-Curve. *SIAM Review* **34**(4), 561–580 (1992)
  21. Marozzo, F., Bessi, A.: Analyzing polarization of social media users and news sites during political campaigns. *Social Network Analysis and Mining* **8**(1) (2018). <https://doi.org/10.1007/s13278-017-0479-5>
  22. Marozzo, F., Rodrigo Duro, F., Garcia Blas, J., Carretero, J., Talia, D., Trunfio, P.: A data-aware scheduling strategy for workflow execution in clouds. *Concurrency Computation* **29**(24) (2017). <https://doi.org/10.1002/cpe.4229>
  23. Marozzo, F., Talia, D., Trunfio, P.: A cloud framework for parameter sweeping data mining applications. pp. 367–374 (2011). <https://doi.org/10.1109/CloudCom.2011.56>
  24. Marozzo, F., Talia, D., Trunfio, P.: A cloud framework for big data analytics workflows on azure. *Advances in Parallel Computing* **23**, 182–191 (2013). <https://doi.org/10.3233/978-1-61499-322-3-182>
  25. Marozzo, F., Talia, D., Trunfio, P.: Js4cloud: Script-based workflow programming for scalable data analysis on cloud platforms. *Concurrency Computation* **27**(17), 5214–5237 (2015). <https://doi.org/10.1002/cpe.3563>
  26. Marozzo, F., Talia, D., Trunfio, P.: A workflow management system for scalable data mining on clouds. *IEEE Transactions on Services Computing* **11**(3), 480–492 (2018). <https://doi.org/10.1109/TSC.2016.2589243>
  27. Rodrigo Duro, F., Marozzo, F., Garcia Blas, J., Talia, D., Trunfio, P.: Exploiting in-memory storage for improving workflow executions in cloud platforms. *Journal of Supercomputing* **72**(11), 4069–4088 (2016). <https://doi.org/10.1007/s11227-016-1678-y>
  28. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: *Mass storage systems and technologies (MSST)*, 2010 IEEE 26th symposium on. pp. 1–10. IEEE (2010)
  29. Talia, D., Trunfio, P., Marozzo, F.: *Data Analysis in the Cloud: Models, Techniques and Applications* (2015). <https://doi.org/10.1016/C2014-0-02172-7>
  30. White, T.: *Hadoop: The definitive guide.* ” O’Reilly Media, Inc.” (2012)
  31. Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., et al.: Apache spark: A unified engine for big data processing. *Communications of the ACM* **59**(11), 56–65 (2016)