Meta-learning in Grid-based Data Mining Systems

Moez Ben Haj Hmida and Yahya Slimani

Faculty of Sciences of Tunisia
Campus Universitaire. 2092 El Manar, Tunis, Tunisia
Phone : +216 71 872 600 - Fax : +216 71 871 666

{moez.benhajhmida, yahya.slimani}@fst.rnu.tn

Abstract

The Weka4GML framework has been designed to meet the requirements of distributed data mining. In this paper, we present the Weka4GML architecture based on WSRF technology for developing meta-learning methods to deal with datasets distributed among Data Grid. This framework extends the Weka toolkit to support distributed execution of data mining methods, like meta-learning. The architecture and the behaviour of the proposed framework are described in this paper. We also detail the different steps needed to execute a meta-learning process on a Globus environment. The framework has been discussed and compared to related works.

Keywords

Data mining; meta-learning; grid computing; distributed dataset; WSRF.

1 Introduction

Nowadays, we have a deluge of data from scientific, industry and commerce fields. Massive amounts of data that are being collected are often heterogeneous, geographically distributed and owned by different organisations. These data contain hidden knowledge that have to be extracted. Data mining (also known as Knowledge Discovery) is the process of discovering useful knowledge from data.

Data mining is a massive computing task that deals with memory resident data. With the huge amount of stored data in a centralised or distributed system, traditional data mining techniques are inefficient. The need of parallel and distributed computing becomes inevitable to deal with large-scale data mining (Kargupta, 2000). Parallel data mining deals with tightly-coupled systems based on fast networks, while distributed data mining deals with loosely-coupled systems with lower interconnection. A wide range of algorithms have been proposed for these two fields and demonstrate relatively good performances with large-scale datasets. But with the emerging concept of Virtual Organizations that proposes to interact different administration domains or organizations as a unified system, these techniques become obsolete. Obviously the design of new solutions based on grid systems is needed.

The Grid is the computing architecture that provides capabilities for manipulating geographically distributed resources as a single meta-system. The Grid provides protocols and services to share computing resources data and software managed by multi-institutional virtual organizations (Reed, 2003). Grid services are provided by middleware systems such as Globus, Legion, gLite, and Unicore (Reed, 2003). These middleware systems are meta-systems that run on top of existing operating systems and networks. The development of grid systems supporting data mining is needed to analyse and extract knowledge from data stored on grid systems. The design of such systems has to take into account emerging Grid standards like Web Service Resource Framework (WSRF).

WSRF is a set of proposed Web services specifications that describe the means by which a *stateful resource* is defined and associated with a Web services description, forming the overall type definition of a WS-Resource (Czajkowski, 2004).

This paper describes the development of a new framework based on WSRF standards and discusses design aspects and execution mechanisms. This framework, called Weka4GML, aims to port the Weka (Witten, 2005), a widely used data mining suite, to a Grid environment.

The remainder of the paper is organised as follows: section 2 gives a background about the learning classifiers from distributed data sources, where section 3 introduces the evolution of distributed data mining to Grid environments. Section 4 presents a WSRF-based implementation of the Weka4GML services. Section 4 tries to compare our proposition to previous works. Finally, Section 6 concludes the paper and opens future perspectives.

2 Distributed data mining

The objective of distributed data mining is to perform the data mining operations using the peculiarities and the availability of distributed resources (Kargupta, 2000). It deals with loosely-coupled systems: generally share-nothing machines interconnected by high-latency networks and geographically distributed (Zaki, 2000).

Typically a distributed data mining algorithm partitions the database at several sites and mines a local model for each partition, then merges them to build a global model. DDM algorithms try to learn from the distributed data generally without exchanging the raw data.

Most of the distributed data mining classification approaches are based on the *ensemble learning* method (Dietterich, 2000; Promidis, 2000). Ensemble methods are learning algorithms that construct a set of classifiers (base-classifiers) and then classify new data points by taking a (weighted or unweighted) vote of their predictions (Dietterich, 2000). Typically, an ensemble learning method runs the learning algorithm several times, each time with a different subset of the training examples. This method can be applied to a distributed environment, where data are stored on different sites, thus producing different local models to be aggregated into a global model. Based on this observation, several methods have been proposed such as: meta-learning, knowledge probing, and collective learning (Kargupta, 2000).

The *meta-learning* method was proposed for homogeneous distributed databases (Prodromidis, 2000). A meta-learning process follows three steps, as shown in figure 1:

1 A base-classifier is trained from the local training-set at each site.

2 The produced classifiers are collected to a central site where their predictions are generated on a separate validation-set to produce metalevel data.

3 The global classifier (meta-classifier) is trained from the meta-level data.

The meta-classifier algorithm can be an *arbiter* or a *combiner* of the different predictions composing the meta-level data. Meta-learning reduces the communication cost when it exchanges the learned models instead of training examples and shows good scalability on data size when it learns from small subsets that fit in memory. These properties make the meta-learning method suitable for large systems like Grids.

3 Grid-based data mining

Grid computing is a natural evolution of distributed computing that attempts to better utilize unused compute capacity by exploiting the computing power of a large numbers of server computers, desktop PCs, clusters, and other kind of hardware. The Grid is a distributed infrastructure that coordinates resource sharing and problem solving in dynamic, multi-institutional virtual organizations. A virtual organization consists on a set of individuals and institutions sharing resources regarding a set of rules (Reed, 2003).

Traditional distributed computing technologies do not provide the flexibility and control on sharing relationships needed to establish VOs. The Open Grid Services Architecture (OGSA) (Reed, 2003) is proposed as a Service Oriented Architecture (SOA) for the development of Grid systems. OGSA defines the mechanisms for creating, managing, and exchanging information among entities, called Grid Services. The Globus Toolkit is a middleware providing a set of OGSA capabilities based on WS-Resource Framework (WSRF) (Czajkowski, 2004). The WSRF is a set of Web Service specifications that describe how to implement OGSA capabilities using Web services.

Grid-based data mining has emerged for supporting complex data mining operations over geographically distributed large data sets. Grid systems provide services for data access and management for such distributed data. However, these services are not sufficient to analyze and extract knowledge from huge amount of data stored within grid environments. Efforts for Grid-enabling data mining applications produced a variety of systems, algorithms, and other tools (Ben HajHmida, 2009).

Several projects like TeraGrid (Catlett, 2007) focused on the creation of Grid infrastructure providing tools for the management and the manipulation of distributed data sources. Other projects like the Knowledge Grid (Cannataro, 2003) have focused on the creation of systems for distributed knowledge discovery over Grid infrastructures.

The *TeraGrid* project aims to provide a Grid infrastructure (Catlett, 2007) built over resources available at four main sites (San Diego Supercomputing Center, National Center for Supercomputing Applications, Caltech and Argonne National Lab). The architecture of the TeraGrid adopts the existing Grid software technologies building a "virtual system" that describes the capabilities and behavior of a resource through *service specifications*. TeraGrid offers a set of application services, called *TeraGrid Application Services*, implemented upon basic software components, called *Grid Services.*

The *Knowledge Grid* (Cannataro, 2003) is a high-level, service-oriented framework providing grid-based data mining tools and services. The Knowledge Grid system uses the low-level features of Globus allowing for the creation of geographically distributed data mining applications and related tasks, such as data management and knowledge representation. Recent developments of the Knowledge Grid (Congiusta, 2007) have accomplished a re-design and re-implementation of all the Knowledge Grid services as WSRF-compliant Grid Services. In such latter version, the Knowledge Grid exposes two main set of services: Resource Management Services and Execution Management Services. Through such services it is possible to design and execute complex knowledge discovery applications through a visual environment.

TeraGrid and he Knowledge Grid are the most complete ongoing projects embracing the emerging trends and providing an almost complete set of functionalities. But the deployment and the use of such systems are quite consequent. The execution of a data mining task on distributed datasets needs a design of a complex mining process by an experimented user. Contrariwise we propose a simpler framework that hides to the user the complexity of distributed data mining tasks execution.

4 Weka4GML framework

In this section we propose a novel framework, Weka4GML, enabling the support of meta-learning on a Grid environment. Based on Web services technology, Weka4GML extends the Weka toolkit (Witten, 2005) to Grid environments. Weka is a collection of sequential data mining algorithms for knowledge discovery comprising standard data preprocessing, mining, and visualization techniques. The Weka4GML framework is based on the Globus Toolkit, a widely used Grid middleware supporting the WSRF standard. The proposed framework architecture is WSRF compliant to be deployed on top of the Globus middleware.

4.1. Architecture description

The proposed architecture is designed according to the meta-learning execution scheme. As shown in Figure 2, our framework is composed of four node types: *storage node*s, *base-classification nodes*, *metaclassification nodes* and *user nodes*. A storage node hosts one or several fragments of the distributed dataset. It also publishes the stored data and their properties as a Web service and shares local data with other Grid nodes via an FTP server. A base-classification node builds a local model by executing a base-classification algorithm on a local dataset partition. The produced model is then applied on the validation dataset to generate predictions and transfer them to the meta-classification node. A metaclassification node integrates an FTP server to collect meta-data on which a meta-level algorithm is executed to produce the final classifier. A user node offers a graphical interface allowing users to choose the Weka supported algorithms, to explore the datasets stored on the storage nodes, and to execute a meta-learning process on a Grid.

4.2. Behaviour description

As shown in Figure 2, each node comprises a Web service, allowing remote applications invocation. Depending on its type, a node can also comprise a Weka module, an FTP server, or a database server. In the following, we provide a behaviour description of the different nodes composing the framework.

Storage node: a storage node is composed of a database server, an FTP server and a Web service (*Data Service*). The database server stores the datasets to be mined. If the dataset is distributed on several partitions, they can be stored on one storage node or among several ones. The FTP server contributes to the data distribution management, allowing the dataset partitions transfer to other storage nodes (validation data) or a baseclassification node. The Data Service interacts with the data management service of a user node by exposing properties concerning the stored datasets. These properties, described in an XML file, comprise the dataset name, its type (centralised or distributed), its size and the data type (discrete or continuous).

Base-classification node: a base-classification node is composed of a Weka module and a Web service (*Base-Classification Service*). The Weka module comprises the set of algorithms provided by Weka toolkit. These algorithms are exposed by the base-classification service allowing remote invocation. The Base-Classification Service acts as an FTP client when it downloads from a storage node the data to be analysed. It allows also meta-data upload towards meta-classification node.

Meta-classification node: a meta-classification node differs from a base-classification node by including an FTP server. This kind of node allocates base-classification tasks to the base-classification nodes participating to the meta-learning process, and then collects the meta-data produced by these nodes. Since a meta-classification node comprises a Weka module, it can also act as a base-classifier. In addition of exporting the algorithms provided by Weka module, the *Meta-Classification Service* notifies the user node of the final result.

User node: a user node is composed of a Weka module, a data manager and a meta-learning module. The Weka module allows the user to choose the base-classification and meta-classification algorithms composing a meta-learning process. The data manager allows gathering the properties of the datasets stored on the Grid. These properties are used to identify the various datasets and their distribution schema (number of partitions, addresses of the storage nodes, etc). After choosing the algorithms and the dataset to be analysed, the user submits a meta-learning process to the meta-learning module. This module interacts with the Meta-Classification Service, to submit the needed requests to the execution of the metalearning process. Each request contains the meta-classifier name, the baseclassifiers names, the base-classification node URIs and URLs of the respective dataset partitions to be analysed.

4.3. Web Service Operations

As recommend by the WSRF specifications, the *WS-ResourceLifetime* supplies some basic mechanisms to manage a resource lifecycle. This implies three basic operations: resource creation (*createResource*), subscription to notification (*subscribe*), and resource destruction (*destroy*).

Meta-learning services

In addition of these WSRF specific operations, each Weka4GML service needs another operation to execute the meta-learning process:

- Meta-Learning Service exposes *metaLearn* operation through which it receives the arguments of the process to be executed.
- Base-Learning service exposes the *learn* operation which executes a learning algorithm on target data.

The arguments needed by Meta-Learning Service to execute the *metaLearn* operation are declared on the WSDL file as *metaLearningParameters*. Figure 3 shows an excerpt of the WSDL type declarations associated with the Meta-Learning Service. The *algorithm* type defines data mining algorithm identified by its name and parameters, where the *algorithmList* type is a

complex type defining a list containing at least one element. In the same way, a type *datasetList* is defined as a list containing at least one dataset. The *dataset* type contains a name describing the dataset, a boolean telling if the dataset is distributed or not and an *uri* permitting the remote access to the dataset partition. The *datasetList* describes all partitions forming the distributed dataset.
The meta-learning process is performed through the *metaLearn* operation starting from the parameters passed by the client. These parameters contain the list of dataset partition, their relative validation set, the list of base-learning algorithms and the meta-learning algorithm.

Whenever the Meta-learning service is invoked, it creates a new WSResource which will store the result of the meta-learning process on the *FinalModel* property. This property is declared on the WSDL as shown by Figure 4.

Data services

In addition of these WSRF specific operations, a *data service* of a storage node needs other operations to handle the stored data Data Service
exposes:
- *createData* operation to store datasets or dataset partitions. This operation is invoked by a user node to add a new distributed dataset among storage nodes.
- *searchData* operation to search stored datasets on grid nodes. This operation is invoked by a user node to search available datasets.
- *createValidationData* operation to create a validation dataset that will be used on the meta-learning process. This operation is invoked by a user node after adding a new dataset or by the meta-learning service during a meta-learning process execution.

The arguments needed by Data Service to execute the *createValidationData* operation are declared on the WSDL file excerpt shown in Figure 5. The sampling algorithm is responsible of extracting a number of representative rows of each data partition contained by the *datsetList*. In each concerned storage node, local validation dataset is created. Finally, local datasets are exchanged between storage nodes to be merged to build the validation dataset to be used on the meta-learning process.

5 Meta-Learning process execution

This section describes the various steps of a meta-learning task execution in Weka4GML framework. Figure 5 describes the interactions between the different framework nodes and their respective Web services for the execution of a meta-learning task. This example does not take into account the dataset discovery by the data management module (user node) as well as the creation of validation data by the storage nodes. The example supposes that the data mining process was already submitted to the meta-learning module. This process is executed by respecting the following steps (see Figure 6):

1 *Resource creation* (meta-classification node): the meta-learning module (user node) invokes the *createResource* operation of the Meta-Classification Service, which creates a new WS-Resource. This resource will be in charge of the maintenance of the state of the submitted meta-learning process. The state is stored as *final model* resource property. This resource is identified by a unique reference, *EndPoint Reference* (EPR), that distinguishes it from the remainder of

the resources of the same service. This reference will be used, thereafter, by the meta-learning module for future invocations of this resource.

2 *Notification subscription* (meta-classification node): the meta-learning module invokes the *subscribe* operation, which subscribes to the notification of the changes of *final model* resource property. With each change of this property, one notification containing the new value will be sent to the meta-learning module.

3 *Task submission* (meta-classification node): the meta-learning module invokes the *metaLearn* operation, which executes a meta-learning algorithm. This operation receives arguments such as the metaclassifier name, the names of the base-classifiers and the baseclassification node URIs and URLs of the partitions to be analysed.

4 *Resource creation* (base-classification node): the Meta-Classification Service invokes in turn the *createResource* operation of the Base- Classification Services passed as arguments in the *metaLearn* operation. In each node, a new WS-Resource is created and identified by an EPR.

5 *Notification subscription* (base-classification node): the Meta- Classification Service node subscribes to the notification of the changes of the *predictions* property of each new resource of a Base- Classification Service.

6 *Task submission* (base-classification): the Meta-Classification Service invokes the *learn* operation of each Base-Classification Service to execute base- classification algorithm in parallel. The *learn* operation receives arguments such as the base-classification algorithm name and the URL of the partition to be analysed.

7 *Data transfer*: each Base-Classification Service downloads the dataset partition to be analysed and the corresponding validation data, by sending a request to the FTP server of the target storage node.

8 *Base-classification*: when data to be analysed are downloaded, each node of the Base-Classification Service executes the required algorithm. A classification algorithm is executed to create a basemodel, tested on the validation data, to generate predictions which will form the meta-data. The result is recorded in the *predictions* property of the resource created on step 4.

9 *Results notification* (base-classifiers): whenever the state of the property *predictions* has been changed, the Base-Classification Service notifies its new value to the Meta-Classification Service by invoking the *deliver* operation. A request is sent to the FTP server of the meta-classification node to transfer the validation data. Since the validation data is the same on all nodes, the FTP server handles the first transfer request and ignores the others.

10 *Resource destruction* (base-classification node): the Meta- Classification Service invokes the *destroy* operation whenever it is notified by a Base- Classification Service, which destroys each resource created on step 4.

11 *Meta-classification*: upon predictions of the various base-classifiers and the gathered validation data, the Meta-Classification Service executes a meta-level algorithm provided by the Weka module to create the meta-data and to generate the final model. The result is stored in *final model* property of the resource created on step 1.

12 *Final result notification*: when the state of *final model* property has been changed, the Meta-Classification Service notifies its new value to the user node by invoking the *deliver* operation.

13 *Resource destruction* (meta-classification node): the user node invokes the *destroy* operation, which destroys the resource created on step 1.

6 Related works

Many projects like *GridWeka* (Khoussainov, 2004), *WekaG* (Pérez, 2005), *Weka4WS* (Talia, 2005) and *FAEHIM* (Ali, 2005) aimed to adapt the toolkit Weka to a Grid environment.

GridWeka (Khoussainov, 2004) is an ongoing work at the University of Dublin, which distributes classification algorithms using the crossvalidation method over computers in an ad-hoc Grid. The system is composed of two main components: *Weka Server* and *Weka Client*. Each participating machine runs the original Weka as server component. The Weka client allows users to specify the target algorithms and datasets, and also to specify resources and tasks constraints.

Another system using the client server architecture is *WekaG* (Pérez, 2005). The WekaG server implements data mining algorithms while the client side instantiates grid services. WekaG is an implementation of a more general architecture called *Data Mining Grid Architecture* (*DMGA*) based on Weka and Globus Toolkit 3. DMGA provides the composition of services on workflows. WekaG provides the same user interface as Weka and can support parallel algorithms. The first prototype implemented uses only the Apriori algorithm as a grid service. Unfortunately, this algorithm cannot be used by meta-learning methods, whereas Weka4GML allows the use of any algorithm supported by Weka by exposing the entire Weka module as a Web service.

Weka4WS (Talia, 2005) extends Weka allowing the execution of all its data mining algorithms on remote Grid nodes. To enable remote invocation, the data mining algorithms provided by the Weka library are exposed as a Web Service, which can be easily deployed on the available Grid nodes. The architecture of Weka4WS includes three kinds of nodes: *storage nodes*, which contain the datasets to be mined; *compute nodes*, on which remote data mining algorithms are run; *user nodes*, which are the local machines of users. Remote execution is managed using basic WSRF mechanisms (state management, notifications, etc.), while the Globus Toolkit 4 services are used for standard Grid functionalities, such as security and file transfer. Weka4WS can only handle a dataset contained by a single storage node. This dataset is then transferred to computing nodes to be mined. If data are considerably large this transfer will cause high communication overhead. On the contrary, Weka4GML handles naturally distributed datasets that are contained by several storage nodes. In its last version Weka4WS supports the design of complex discovery processes, such as meta-learning, by porting the Weka KnowledgeFlow to Web services. This means that the user has to compose by itself all the meta-learning process, whereas Weka4GML takes in charge the process composition making it transparent to the user.

Very similar is the Federated Analysis Environment for Heterogeneous Intelligent Mining (*FAEHIM*)) project (Ali, 2005), aimed to support data mining based on a grid services approach. The data mining algorithms implementations are derived from the Weka library and converted into a set of grid services. Complex mining processes are designed using the Triana (Churches,

2005) workflow engine. Like Weka4WS, FAEHIM do not support the data distribution and leaves to the user the charge of complex discovery processes design, whereas Weka4GML supports data distribution and automates the composition and the execution of a metalearning process.

GridWeka and WekaG tried to adapt Weka to a Grid infrastructure using the client server architecture. But this architecture is not so flexible and powerful to support large scale requirements such as security, resource management etc. By contrast, Weka4GML, Weka4WS and FAEHIM are based on a grid services approach.

Compared to the existing systems, our framework deals with distributed data and offers an easier execution of the meta-learning process. Indeed, Weka4WS and FAEHIM provide workflow engines by which users can compose a complex workflow schema such as the distribution of a dataset on different nodes and the execution of a meta-learning process. However, Weka4GML offers to users to discover distributed datasets, to choose the base classifiers to use, and then the framework holds the execution and the coordination of the distributed tasks on the target nodes.

Table 1 compares the reviewed works and the Weka4GML framework according to the system architecture, the supported standards, the used middleware, and the support of workflow. In this table we note Middleware by MW, Web Services by WS and Extended Weka workflow engine by EWWE.

Table 1 Comparing Weka4GML to the reviewed frameworks

System	Architecture	Standard	MW	Workflow
GridWeka	Client/ server	WS	JVM	BPE
WekaG	Client/ server	WSRF	Globus3	Ad-hoc
Weka4WS	Grid services	WSRF	Globus4	EWWE
FAEHIM	Grid services	WS	Globus4	Triana
Weka4GML	Grid services	WSRF	Globus4	Framework handled

7 Conclusion

In this paper, we described the evolution of data mining from distributed systems to Grid systems. Grid computing is a promising architecture that opens new horizons and faces new challenges. We also proposed a new framework based on WSRF technology for the distributed execution of the meta-learning approach. This framework is deployed on the top of the Globus middleware. Comparatively to similar works, Weka4GML is a framework which allows the discovery of knowledge in a completely distributed manner, starting from distributed data.

The proposed framework can not concurrent other systems like TeraGrid or the Knowledge Grid. It is considered as an initiative to port parallel and distributed mining algorithms to Grids regarding the peculiarities of such infrastructure.

As perspective of our work, we propose to extend this framework to support other distributed data mining techniques and to manage faulttolerance.

References

Ali, A. S., and Taylor, I. J. (2005) 'Web services composition for distributed data mining', *In the 2005 IEEE International Conference on Parallel Processing Workshops*, pp. 11-18.

Ben HajHmida, M., Congiusta, A. (2009) 'Parallel, Distributed, and Grid-Based Data Mining: Algorithms, Systems, and Applications', in Mario Cannataro (Ed.), *Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine and Healthcare*, IGI Global.

Cannataro, M. and Talia, D. (2003) 'The knowledge grid: An architecture or distributed knowledge discovery', *Commun. ACM*, *46*(1), pp. 89-93.

Catlett, C. et al (2007), 'TeraGrid: Analysis of Organization, System Architecture, and Middleware Enabling New Types of Applications', In L. Grandinetti (Ed.), *Advances in Parallel Computing*. IOS Press.

Churches, D., Gombas, G., Harrison, A., Maassen, J., Robinson, C., Shields, M., Taylor, I., and Wang, I. (2005) 'Programming scientific and distributed workflow with triana services', *Concurrency and Computation: Practice and Experience*, *18*(10), 1021-1037.

Congiusta, A., Talia, D., and Trunfio, P. (2007) 'Distributed data mining services leveraging wsrf', *Future Generation Computing Systems*, *23*(1), 34-41.

Czajkowski, K. (2004) '*The WS-Resource Framework Version 1.0'*, http://www-106.ibm.com/developerworks/library/ws-resource/wswsrf.pdf.

Dietterich, T. G. (2000) 'An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization', *Machine Learning*, *40*, 139-157.

Kargupta, H., and Chan, P. (Eds.), (2000). *Advances in Distributed and Parallel Knowledge Discovery*. MIT Press.

Khoussainov, R., Zuo, X., and Kushmerick, N. (2004) 'Grid-enabledweka: A toolkit for machine learning on the grid', *ERCIM News*, *59*.

Pérez, M. S., Sànchez, A., Herrero, P., Robles, V., and Pena, J. M. (2005) 'Adapting the weka data mining toolkit to a grid based environment', *In 3rd Atlantic Web Intelligence Conference,* pp. 492-497.

Prodromidis, A. and Chan, P. (2000) 'Meta-learning in Distributed Data Mining Systems: Issues and Approaches', In Hillol Kargupta and

Philip Chan (Eds.), *Advances of Distributed Data Mining*. MIT/AAAI Press, pp. 81-114.

Reed, D. A., Mendes, C. L., Lu, C., Foster, I., and Kesselman, C. (Eds.), (2003) *The Grid 2: Blueprint for a New Computing Infrastructure - Application Tuning and Adaptation*. Morgan Kaufman.

Talia, D., Trunfio, P., and Verta, O. (2005) 'Weka4WS: A WSRF-Enabled Weka Toolkit for Distributed Data Mining on Grids', *In the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, *3721*, 309-320.

Witten, I. H., and Frank, E. (Eds.), (2005) *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Authors

Moez Ben Haj Hmida received the Master degree in computer science from the University of Tunis El Manar, Tunisa, in 2005. He is currently pursuing the PhD in computer sciences at the same university. He is currently lecturer at the department of Computer Science of Faculty of Sciences of Tunis, Tunisia. His current research interests include parallel, distributed and Grid-based data mining applications.

Yahya Slimani received the B.Sc.(Eng.), Dr Eng and PhD degrees from the Computer Science Institute of Alger's (Algeria), University of Lille (French) and University of Oran (Algeria), in 1973, 1986 and 1993, respectively. He is currently Professor at the Department of Computer Science of Faculty of Sciences of Tunis. These research activities concern data mining, parallelism, distributed systems and grid computing.
Dr. Slimani has published more than 80 papers from 1986 to 2005. He contributed to Parallel and Distributed Computing Handbook, Mc Graw-Hill, 1996. He joined the Editorial Boards of the Information International Journal in 2000.

Figure 1 Meta-learning from distributed data sources

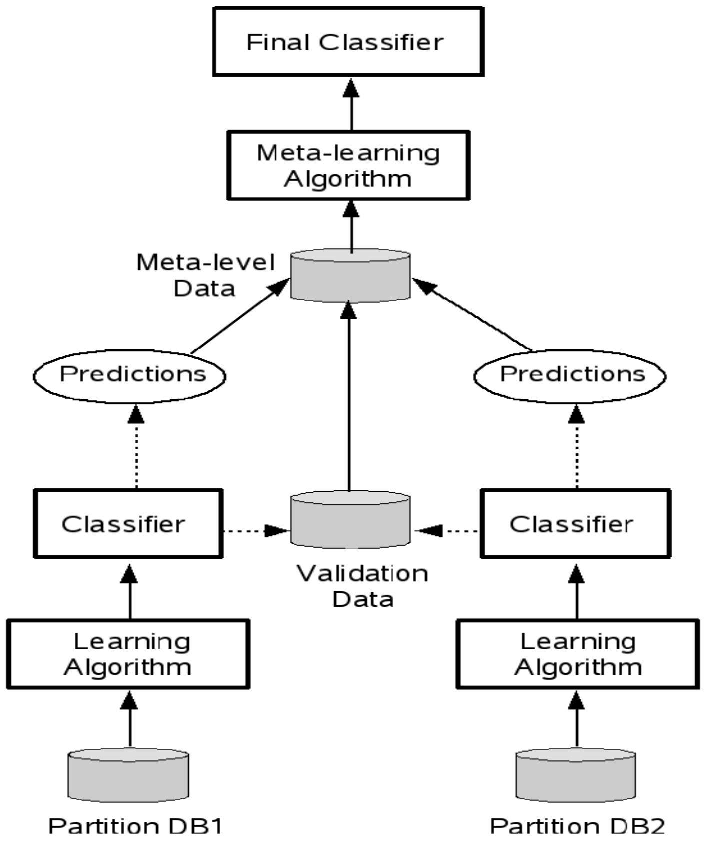


Figure 2 Meta-learning process on Weka4GML framework

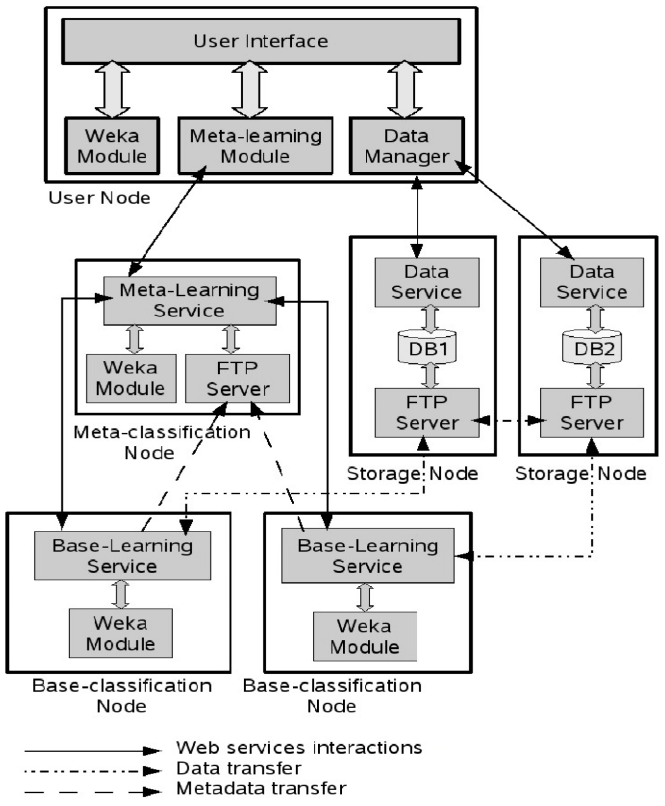


Figure 3 Input/Output parameters of the *metaLearn* operation

```
<types>
...
<xsd:element name="algorithm">
  <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="name" type="xsd:string"/>
     <xsd:element name="parameters" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name=" algorithmList ">
```

```
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name="algorithm" type="xsd: algorithm" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="dataset">
 <xsd:complexType>
   <xsd:sequence>
     <xsd:element name="name" type="xsd:string"/>
     <xsd:element name="distributed" type="xsd:boolean"/>
     <xsd:element name="uri" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<xsd:element name=" datasetList ">
 <xsd:complexType>
  <xsd:sequence>
     <xsd:element name=" dataset " type="xsd: dataset " minOccurs="1"
maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="metaLearningParameters">
 <xsd:complexType>
   <xsd:sequence>
     <xsd:element name="taskID" type="xsd:long"/>
     <xsd:element name="metaLearningAlgorithm" ref="tns:algorithm"/>
     <xsd:element name="baseLearnersList" ref="tns:algorithmList"/>
     <xsd:element name="datasetList" ref="tns:datasetList"/>
     <xsd:element name="validationData" ref="tns:dataset"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</types>
```

Figure 4 Meta-Learning Service Messages

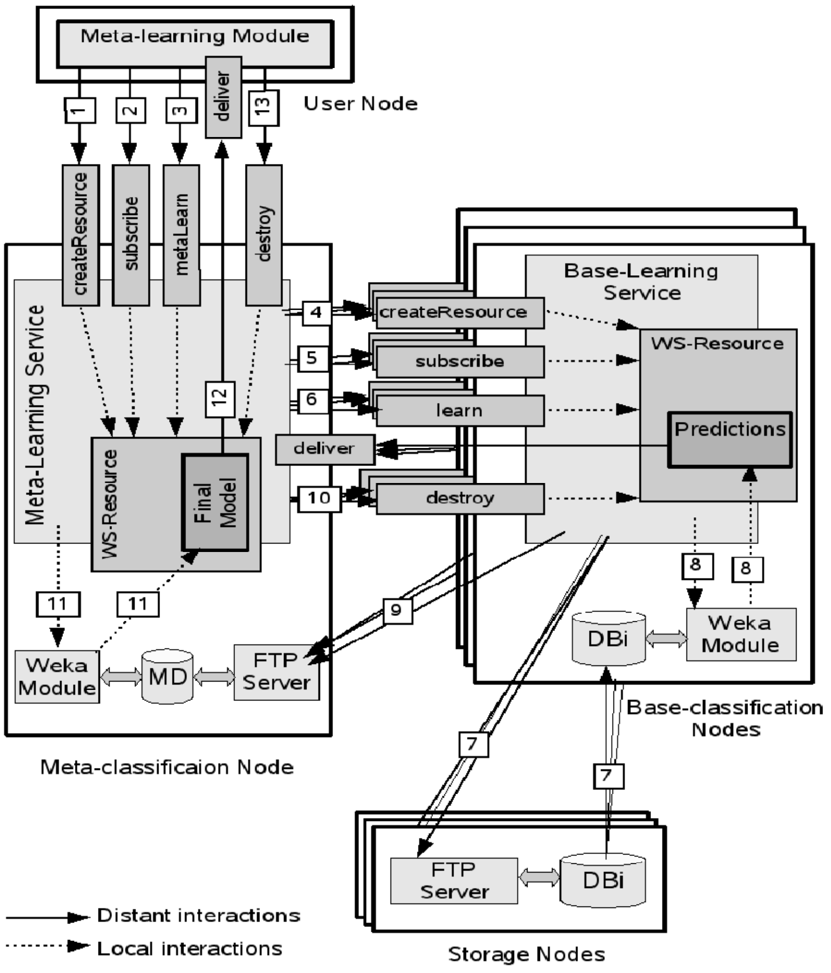
```
          <!-- RESOURCE PROPERTIES -->
<xsd:element name="FinalModel" type="xsd:string"/>

<xsd:element name="MetaLearningResourceProperties">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns: FinalModel " minOccurs="1"
        maxOccurs="1"/>
     </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Figure 5 Input/Output parameters of the *createValidationData* operation

```
<xsd:element name=" createValidationDataParameters">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="taskID" type="xsd:long"/>
      <xsd:element name="samplingAlgorithm" ref="tns:algorithm"/>
      <xsd:element name="datasetList" ref="tns:datasetList"/>
      <xsd:element name="validationData" ref="tns:dataset"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 6 Execution steps of meta-learning process


Meta-learning Module
User Node
1
2
3
deliver
13
createResource
subscribe
metaLearn
destroy
Meta-Learning Service
WS-Resource
Final Model
12
11
11
Weka Module
MD
FTP Server
Meta-classificaion Node
4
5
6
10
createResource
subscribe
learn
deliver
destroy
Base-Learning Service
WS-Resource
Predictions
9
8
8
DBi
Weka Module
Base-classification Nodes
7
7
FTP Server
DBi
Storage Nodes
Distant interactions
Local interactions