CDM: A Prototype Implementation of the Data Mining JDM Standard

Piotr Lasek

Chair of Computer Science, University of Rzeszów
ul. Prof. St. Pigonia 1, 35-310 Rzeszów, Poland
lasek@ur.edu.pl

Abstract. There exists a great variety of tools and applications designed for data mining and knowledge discovery. Historically, from the 1970s, a number of available tools continues to grow. For this reason, a potential user may have difficulties when trying to choose an appropriate tool for himself. Similarly, when it comes to the implementation and evaluation of newly proposed data mining algorithm, an author needs to consider how to verify his proposal. Usually, a new algorithm or a method is implemented and tested without using any standardized software library and tested by means of an ad hoc created software. This causes difficulties in case when there is a need to compare efficiency of two methods implemented using different techniques. The aim of the paper is to present a prototype implementation of a data mining system (CDM) based on the Java Data Mining standard (JDM) that provides standardized methods designed for convenient implementation and verification of data mining algorithms.

Keywords: Data mining, clustering, JDM standard, CDM.

1 Introduction

Data mining is a relatively new and rapidly developing field of computer science. Its main goal is to explore data so that new, unknown and potentially useful patterns could be discovered. Data mining methods employ different and specialized algorithms for building, modeling and evaluation of discovered knowledge and are used to analyze multiple kinds of data from many domains such as medicine, healthcare, finance, telecommunication, science, etc. Recently, data mining tools, crucially improved and simplified data mining by employing new efficient algorithms [3, 4, 6]. The number of methods and tools both commercial and open source increases rapidly every year [2, 5, 7]. Undoubtedly, data mining tools and applications become widely known and used when it comes to exploring knowledge from large amounts of data.

Data mining techniques evolve so that they become both more expert and problem oriented [14]. Additionally, during recent years, they also became more useful in everyday applications, such as, for example, e-mail filtering or credit card fraud analysis [15]. For professionals, the data mining process is more like an art because of the fact that they usually limit their analysis to several best known techniques. On the other

W. Zamojski et al. (eds.), *Proceedings of the Ninth International Conference DepCoS-RELCOMEX*, Advances in Intelligent Systems and Computing 286,
DOI: 10.1007/978-3-319-07013-1_29,

hand, beginners are often overwhelmed by the variety of methods and the diversity of existing different, both commercial and open source, data mining tools. The diversity of these tools was a subject of numerous works [2]. The aim of our work is not to compete with existing software but rather to present and promote the way how to implement and test data mining algorithms so that they could be easily reused by another users.

Among numerous tools and data mining libraries the introduction of the Java Data Mining standard (JDM) is a step towards standardization and vendor neutral development of data mining solutions. JDM was designed so that it is based on solid concepts such as so-called mining objects, models and tasks. On the other hand, despite it comes with standard, its Application Programming Interface (API) is also flexible and extensible.

The paper is divided into four sections. After the introduction we recall basic definitions of terms used in the next part of the paper which are related to the implementation of the prototype CDM (for *Common Data Mining*) system based on the JDM standard. Additionally, in the second section we briefly describe the JDM standard and present, from our perspective, its crucial features. In Section 3 we describe our prototype implementation of data mining engine and several other components required to meet the standard of JDM. We summarize our paper as well as present our further plans related to development of CDM in Section 4.

2 The Architecture of JDM

JDM was created by high-class experts and meets the following crucial assumptions: addresses a large developer community, is a standard interface, has a broad acceptance among vendors and consumers, is extensible, simplifies data mining for novices while allowing control for experts, recognize conformance limitations for vendor implementations, supports requirements of real, industrial applications, appeals to vendors and architects in other development domains [3]. For these reasons we have chosen JDM as a base for implementation of our prototype CDM.

The specification of JDM was accepted by Java Community Process Executive Committee in 2004. The architecture of JDM comprises three main components, namely: API, DME (Data Mining Engine) and MR (Meta Data Repository). API (Application Programming Interface) is the set of programming interfaces which should be implemented so as to provide access to services available in DME. Data Mining Engine can be implemented in several ways. One possible way is to implement it as a library providing methods to access data to be analyzed, another may be to implement it as a more convenient tool such as a server. In the latter case, such an implementation of DME is usually called DMS (Data Mining Server). Next, the repository of meta data is used to store so-called Data Mining Objects which can be used in different steps of mining process. Repositories can use flat file system or can be programmed as a relational database similarly to the Oracle Data Mining implementation [1]. Moreover, if necessary, it is possible to add additional components that are not included in the specification of the JDM standard.

In Figure 1 we recalled the diagram of several so-called Mining Objects used in JDM. A Mining Object is a base class for all classes in JDM and comprises the basic and common features such as: name, description, identifier or type of an exploration object. A Mining Object can be saved, under a specific name in a Mining Objects Repository and during a mining process, it can be accessed by another methods by using its name. In JDM the data mining objects are divided into the following types: Data Specification Objects, Settings Objects and Tasks. Data Specification Objects are designed for defining input data by using both logical and physical interfaces. For example the Logical Data based classes are used to interpret data whereas Physical Dataset based classes designed to define the place where data are stored as well as the attribute names and data types. By means of these two types of data classes (logical and physical) it is possible to separate data from algorithms. For example, by using Settings Objects, it is possible to provide parameters to data mining functions. There are different kinds of Setting Objects classes which are appropriate for different kinds of mining functions, namely: Clustering Settings, Supervised Settings, Attribute Importance Settings, Association Settings. Settings Objects provide means to control the build and apply process by setting values of processes or algorithms parameters. For example Build Settings based classes are used to specify settings at the function level and optionally at the algorithm level. Apply Settings classes for instance, provide flexibility when defining the results from model apply.

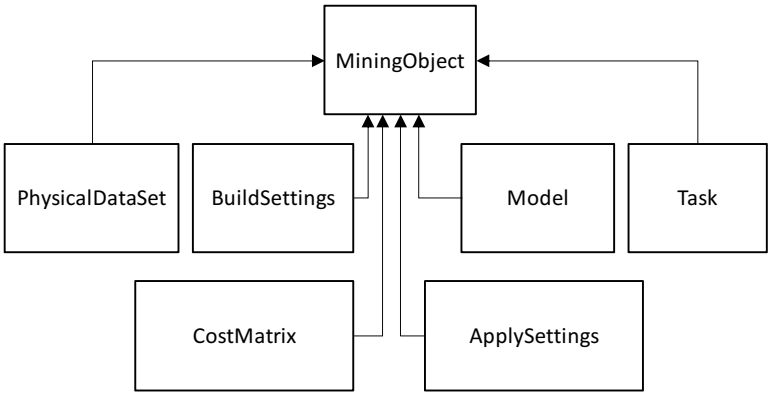

Fig. 1. A simplified class diagram of named object of JDM

Model Objects are used to store a compact representation of the knowledge. This kind of objects provides details at the function and algorithm level. The *Model* interface is the base interface for all models used in JDM. It comprises another interface called *ModelDetail* encapsulating algorithm-specific details. A sample implementation of the model for classification could consist of the following classes *ClassificationModel*, *ClassificationSettings* and *TreeModelDetail*. The first class would provide common content for all kinds of classification algorithms and the *TreeModelDetail* class would provide elements specific to the algorithms based on the decision tree.

Another important element of the JDM architecture is the entity called Task. Tasks represent all information that is required to perform mining operation. Tasks are

started by invoking the *execute* method from the *Connection* object. Because of the fact that when analyzing big data sources mining tasks can be long running, the JDM architecture supports both synchronous and asynchronous execution of tasks. The tasks can be controlled by the handle represented by an object of the *ExecutionHandle* class.

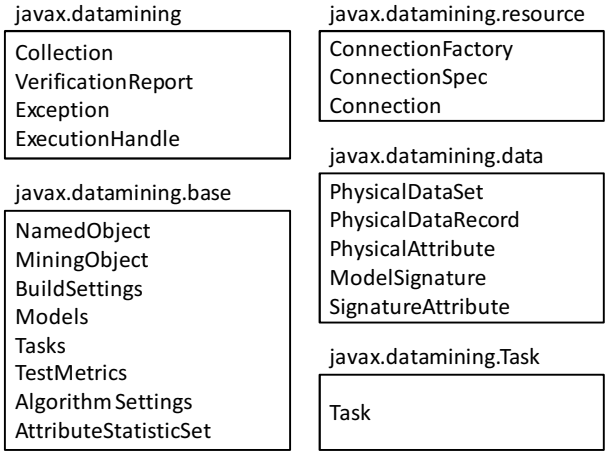

Fig. 2. The set of interfaces (by packages) to be implemented to meet the JDM standard

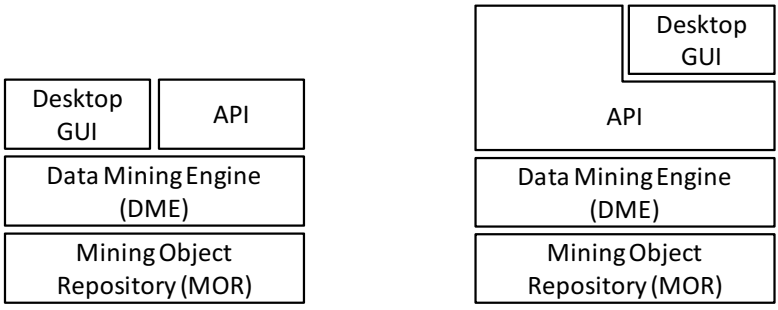

Fig. 3. A sample data mining tool's architecture possible to be implemented using JDM

The JDM architecture allows development of customized implementations of standard interfaces. In order to create an individual implementation based on the JDM standard, the minimum set of interfaces must be implemented. The minimum implementation comprises elements presented in Figure 1. The crucial elements are listed below:

- *Connection*, *ConnectionFactory*, *ConnectionSpec* – these classes are designed to provide access to data and dispatch execution of mining functions according to the specification of the mining experiment.
- *PhysicalDataSet*, *PhysicalDataRecord*, *PhysicalAttribute* – these classes are used to represent datasets as well as data records and attributes. Additionally factories for creation of object of these classes should be implemented. Optionally, if necessary *LogicalData* and *LogicalAttribute* can be implemented.
- Basic implementation of the *Task* interface capable of, at least, synchronous execution of a mining operation with an accompanying factory class to create objects of concrete tasks classes.

In Figure 3 we have recalled two possible architectures of an implementation of the JDM standard to show that the standard is flexible and extensible. For example, when the system is created following the architecture presented in Figure 3a, a user is able access the data mining engine (DME) directly (via graphical user interface) or by means of API. On the other hand, as presented in Figure 3b, the system can be programmed so that the data mining engine cannot be accessed directly, but by public API. Vendors implementing their own JDM based systems can select the most appropriate approach from their perspective. For example, the system can read data from files and, on the other hand, can be more database-centric by reading and storing data in a relational database.

The goal of our work was to create a simple tool for performing ad hoc data mining task by providing functions for loading data from files containing sample benchmark data. We decided to use JDM as a standard framework in order to ensure that created methods could be easily executed in different environments supporting the JDM standards. This could give the possibility to implement and test the implemented algorithm using one system and deploy it easily in another one.

3 The Implementation

All interfaces in JDM are defined as a pure Java specification. For this reason all classes implementing JDM interfaces can be programmed also purely in Java. Nevertheless, vendors have possibility to implemented their own methods behind the JDM interface so that any implementation or technology can be used. In other words, vendors have possibility to wrap up any kind of a source code with the JDM interfaces.

In this section we present how we created the prototype implementation of the system based on the JDM interfaces. First of all we determined and created the minimal set of classes to be implemented to meet the minimum implementation of a functional JDM system. This task required over a dozen classes to be created. In Figure 4, on the left side, we presented those classes, however, for the sake of simplicity, some of optional classes were not shown in the figure. On the right side, we shown how to derive own customized classes from the minimum implementation classes in order to create an implementation of a new algorithm, for example. In our case, we implemented within CDM several clustering algorithms, such as widely known *k-Means* [8] and two version of the *NBC* density based algorithm [9]. One can easily notice that in our custom implementation of the clustering module, it was necessary to create a

package called *javax.datamining.clustering* containing classes deriving from classes implemented in the *javax.datamining.data* package and implementing appropriate JDM interfaces. The next step was to create another package intended for the custom implementation of a single clustering algorithm. Inside this package we placed only two classes extending two base classes such as: the *CDMBasicClusteringSettings* class and the *CDMBasicClusteringAlgorithm* class.

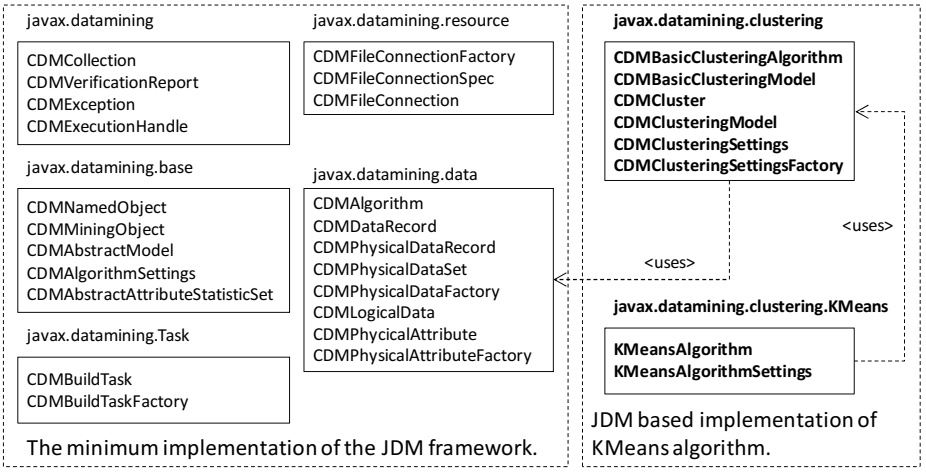

Fig. 4. The prototype implementation (CDM) of the JDM mining system

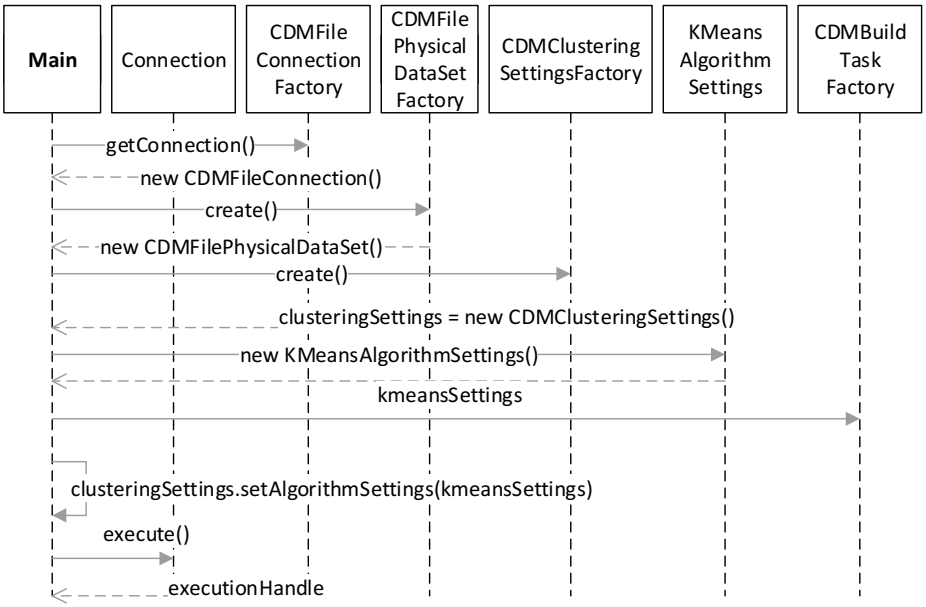

Fig. 5. A simplified sequence diagram presenting the execution of the implemented *k-Means* algorithm

From the perspective of a data mining programmer, when adding a new clustering algorithm to CDM, there are only two classes to be implemented, namely the classes located in a package of a new algorithm. However, if one would like to add an algorithm belonging to the data mining function that is not yet supported by CDM, it would be necessary to create a new package, for example a classification package called *javax.datamining.classification*, and implement the basic classes related to the domain of classification. For example, it might be necessary to create classes such as: *CDMBasicClassificationAlgorithm*, *CDMBasicClassificationModel*, *CDMClass*, *CDMClassificationModel*, etc. – similarly to classes implemented and located in the clustering package.

In Figure 5 we presented the simplified sequence diagram explaining how the sample *k-Means* clustering algorithm is executed. The process of execution of an algorithm comprises several steps. First, it is required to create an object of connection representing a connection to a Data Mining Repository. In our implementation, the connection class was called *CDMFileConnection*. It provides method for reading data from a file system. The path to the source of data (a text file) is given in a form of URI. Then, after creating the connection object using a connection factory, the dataset needs to be prepared. It is done by using *CDMFilePhysicalDataSetFactory* object which creates a *CDMFilePhysicalDataSet* object. Next, co-called physical attributes, are added to the dataset object (this was not shown in the figure) in order to define the structure of the dataset. For example, in this step it is possible to specify which attributes will be taken into account during further experiments and what are the types of those attributes. Attributes, created by means of the *CDMPhysicalAttributeFactory*, are added to the physical dataset. Finally, the defined physical dataset is saved into the Data Mining Object repository by invoking the *saveObject* method which is available in the connection object. The next step is the preparation of the clustering algorithm. Usually every algorithm takes one or more parameters, so at the beginning it will be important to create the settings object (by means of the *CDMClusteringSettingsFactory* class) to make possible to set the appropriate algorithm parameters. The settings factory object returns an object of the *CDMClusteringSettings* class, and then, by using methods provided by the settings object, the user has possibility to set values of parameters of the algorithm. The settings are saved into the data mining repository using the *saveObject* method from the connection object. The last part of the algorithm preparation is the creation of the build task. The build task, represented by an object of the *CDMBuildTask* class, is created by the build task factory (*CDMBuildTaskFactory*). The build task is designed to specify a task that integrates the dataset, the settings (in which the name algorithm used in an experiment is passed) and an output model into which the model of discovered groups will be written. After performing the above actions, namely, specifying the data source, the algorithm and its parameter as well as the output, it is now possible to run the algorithm. It is done by invoking the *execute* method which is provided by the connection object. The execute method returns an execution handle that can be used to control long running task, however, the current implementation of CDM does not provide a possibility to execute long running tasks yet. After the algorithm is ended, it is possible to get results by means of the *retrieveObject* method from the connection object. According to the

specification of JDM, when it comes to clustering, the output of the clustering comprises discovered clusters and rules. So, in the next step it is possible to take advantage of the discovered clusters, for example by verifying or visualizing discovered groups.

4 Conclusions and Further Works

In this paper we have presented the data mining system (CDM) based on the Java Data Mining standard. The interfaces provided by the JDM standard allow creation of the tool supporting all steps of typical process of data mining, such as: data integration, selection, cleaning, integration, data mining, pattern evaluation [13]. CDM covers the minimum implementation required for the system to comply with the JDM standard as well as several clustering algorithms. We consider that using a standardized tool for implementation and testing of new algorithms gives an opportunity to all interested data mining programmers to create software so that it can be easily used and tested in another projects.

The source code of the implemented system is currently available under the following location: http://rspn.univ.rzeszow.pl/?p=682. It can be downloaded using any SVN client. The source code is available in the form of a project of the *Eclipse* platform and it can be run and debugged.

Since our interests are mostly focused on clustering methods, in the nearest future, we will undoubtedly extend our implementation by adding to it more clustering algorithms such as *DBSCAN* [10], *TI-DBSCAN* [11] and *TI-NBC* [12], as well as constrained versions of these algorithms. In cooperation with other authors another missing data mining functions will be gradually added to system.

References

1. Oracle Text. Oracle Text Application Developer's Guide 1Og Release 2
2. Mikut, R., Reischl, M.: Data mining tools. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 1(5), 431–443 (2011)
3. Hornick, M.F., Marcadé, E., Venkayala, S.: Java data mining: strategy, standard, and practice: a practical guide for architecture, design, and implementation. Morgan Kaufmann (2010)
4. Goebel, M., Gruenwald, L.: A survey of data mining and knowledge discovery software tools. ACM SIGKDD Explorations Newsletter 1(1), 20–33 (1999)
5. Kurgan, L.A., Musilek, P.: A survey of Knowledge Discovery and Data Mining process models. Knowledge Engineering Review 21(1), 1–24 (2006)
6. Mariscal, G., Marbán, Ó., Fernández, C.: A survey of data mining and knowledge discovery process models and methodologies. Knowledge Engineering Review 25(2), 137 (2010)
7. Ruotsalainen, L.: Data mining tools for technology and competitive intelligence. VTT (2008)
8. Hartigan, J.A., Wong, M.A.: Algorithm AS 136: A k-means clustering algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics) 28(1), 100–108 (1979)

9. Zhou, S., Zhao, Y., Guan, J., Huang, J.: A neighborhood-based clustering algorithm. In: Ho, T.-B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 361–371. Springer, Heidelberg (2005)
10. Ester, M., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD, vol. 96 (1996)
11. Kryszkiewicz, M., Lasek, P.: TI-DBSCAN: Clustering with DBSCAN by means of the triangle inequality. In: Szczuka, M., Kryszkiewicz, M., Ramanna, S., Jensen, R., Hu, Q. (eds.) RSCTC 2010. LNCS, vol. 6086, pp. 60–69. Springer, Heidelberg (2010)
12. Kryszkiewicz, M., Lasek, P.: A neighborhood-based clustering by means of the triangle inequality. In: Fyfe, C., Tino, P., Charles, D., Garcia-Osorio, C., Yin, H. (eds.) IDEAL 2010. LNCS, vol. 6283, pp. 284–291. Springer, Heidelberg (2010)
13. Han, J., Kamber, M., Pei, J.: Data mining: concepts and techniques. Morgan Kaufmann (2006)
14. Liao, S.-H., Chu, P.-H., Hsiao, P.-Y.: Data mining techniques and applications–A decade review from 2000 to 2011. Expert Systems with Applications 39(12), 11303–11311 (2012)
15. Serban, F., et al.: A survey of intelligent assistants for data analysis. ACM Computing Surveys (CSUR) 45(3), 31 (2013)