The Java Data Mining Package – A Data Processing Library for Java

Holger Arndt
Department of Computer Science
Technical University of Munich
85747 Garching, Germany
Email: arndt@jdmp.org

1 Introduction

Data analysis has become an important task in numerous fields of science as well as in business applications. A number of tools exist for this purpose, some of them specialized for a single task, such as the classification libraries LibSVM or LibLinear, but also general frameworks such as Matlab, Octave or R. Tools for Java are becoming more and more common since they can be used on different operating systems and computer platforms. Some popular Java machine learning packages are Weka, MALLET, KNIME, Java-ML and RapidMiner formerly known as YALE.

However, not all of them are able to handle large data sets, as most of them require to load the data completely into main memory, which becomes impractical at some point. Another limitation is the lack of support for parallel processing in a computer cluster.

In the following sections, I will present the *Java Data Mining Package (JDMP)* [1], an innovative open source data processing library in Java, which has some advantages over existing libraries: It is designed to (1) offer a modular and flexible framework for the representation of algorithms and data sources, (2) allow easy integration of existing data processing libraries, (3) support very large data sets beyond the size limit of main memory, (4) provide mechanism to distribute a task in a computer network, and (5) offer visualization methods to give a better insight on the data.

2 Data Storage and Object Model

The main focus of the library lies on a consistent data representation, using matrices as the basis for all data objects. The *Universal Java Matrix Package (UJMP)* is used for this purpose, which allows to handle very large matrices, which do not fit into main memory. While other packages are limited to two-

dimensional data with less than 2^{32} rows or columns, UJMP can handle very large matrices with up to 2^{63} rows or columns. Dense and sparse matrices as well as multi-dimensional matrices with arbitrary data types such as int, float, or Java objects are supported. Import and export filters are available to attach to various data sources, such as CSV files, images, Excel sheets or SQL data bases. A detailed description of the features can be found in [2]. On top of this matrix library, the here described software introduces a consistent interface hierarchy to represent higher-level objects:

- A number of matrices can be aggregated in a *Variable*, e.g. to store a time series of measurements. The oldest matrix is deleted automatically when a defined capacity limit is reached.
- *Samples* comprise a number of Variables, e.g. "input" and "target" for a classification task. The use of Variables instead of matrices allows to store history which enables the user to observe how classification accuracy evolves for a single sample.
- A *DataSet* is a collection of Samples, e.g. one DataSet for training and one for testing. In addition, a DataSet also contains Variables for measuring e.g. classification accuracy or confusion between classes.
- An *Algorithm* can manipulate Variables, Samples or DataSets. For example, it could read parameters from a Variable and perform classification on a DataSet or a single Sample. The result (e.g. classification error) is written to a Variable which can be used by another Algorithm, to optimize training parameters. An Algorithm can also invoke other Algorithms which provides a convenient mechanism for creating chained calculations.
- A *Module* is a logical entity for grouping objects, e.g. task-specific, per computer, per thread, etc. Modules can also contain other Modules, which allows to create hierarchical or recursive environments.

This model is able to represent various tasks (e.g. classification, optimization, clustering, Bayesian network learning) and data types (e.g. text data, graphical models, ontologies) within the same framework. In particular, the number of Variables in a Sample is not limited and it is possible to include arbitrary information and not only numerical data. In addition, most objects have different representations, e.g. a DataSet can either be accessed as a list of individual Samples, or as a single matrix containing the combined data, which makes it possible to re-use a lot of UJMP's matrix methods within JDMP.

3 Integration of Existing Libraries

As mentioned in the introduction, a number of well-tested machine learning and data processing libraries are available for Java. They provide sophisticated implementations for e.g. clustering or classification algorithms, and it is a

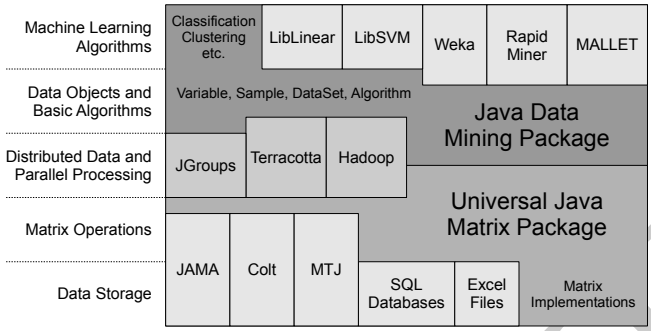

Figure 1: Layer model illustrating the relation between the Java Data Mining Package and other libraries and software tools.

challenging task to offer a new implementation with the same performance and reliability. However, we do not intend to rebuild existing data analysis tools, but rather to provide a general framework to combine them efficiently. To benefit from existing software, a number of interfaces and adaptor classes are provided to facilitate the integration of additional algorithms or storage implementations. As a result, the actual implementation becomes secondary and exchangeable, as long as it can be mapped to the described interface definition. This concept is illustrated in Fig. 1.

In the current stage, adaptors are provided for Weka, MALLET, Lucene, LibSVM, LibLinear, and also for non-Java software such as R, Matlab, Octave and GnuPlot. In the future, we plan to integrate additional libraries and data sources, such as RapidMiner, KNIME, SPSS, SAS and OLAP cubes.

4 Distributed Computing

While the object structure described in section 2 may seem inconvenient at first, its advantages become obvious when it comes to the parallel processing of data. A strict separation of calculation methods (Algorithms) and data structures (Matrix, Variable, Sample, DataSet) is enforced, which is much more rigorous than in other tools and applies also e.g. to the parameters of a learning method or the model itself, which is learned from data. This concept requires to segregate the variables needed for a task from the processing steps and resembles a load-store architecture known from modern CPUs (Variables in our framework are similar to registers of a CPU, they can be discerned by a label and support read and write operations using data in matrix format).

The benefit of this separation is the ability to distribute algorithms and data storage to different computers, and process a task in parallel on several machines. For every data object (e.g. a DataSet) stored remotely on the network, there is an appropriate stub object on the local machine, which redirects all requests to the remote computer accordingly. From a programmer's point of view, it

makes no difference if a Variable or DataSet is remote or not and also the method for sharing data in a network or invoking methods on a remote machine is exchangeable, e.g. transmission of the data could be realized through RMI (remote method invocation), HTTP requests over a SOAP interface, or through common files on a network drive. Another method is the use of additional frameworks such as JGroups, Terracotta, or Apache Hadoop. Since the object model is particularly designed for this purpose making extensive use of Java lists and maps, it is easy to employ methods such as Map-Reduce for processing.

5 Summary and Conclusion

In this paper, I have presented the *Java Data Mining Package (JDMP)*, a universal framework for data processing and machine learning. It facilitates access to data sources and algorithms (e.g. clustering, classification, graphical models, optimization) and provides visualization modules, a description of which has been omitted due to space limitations. The software includes a matrix library [2] for storing and processing arbitrary data types, with the ability to handle very large multi-dimensional matrices even when they do not fit into memory. The main focus of the software lies on a consistent data representation using matrices as the basis for all data structures. Its final goal is to introduce an additional abstraction layer between machine learning tools such as Weka and data processing libraries for parallel computing such as Apache Hadoop, to allow easy deployment in a computer cluster, independent of the technology used for parallel processing (see Fig. 1).

Although it is still in an early stage of development and cannot compete with other libraries in terms of functionality and stability, the framework was able to cope with all our reference implementations. One issue we would like to improve is the availability of documentation, to make it easier to get acquainted with the library. It has to be emphasized, that the Java Data Mining Package is licensed under LGPL, which allows it to be integrated into commercial applications. Source code and jar files are freely available through our website [1] in the hope that it will attract many users and developers.

References

[1] H. Arndt, A. Naegele, and M. Bundschus, "Java Data Mining Package (JDMP)," 2009, http://www.jdmp.org.

[2] H. Arndt, A. Naegele, and M. Bundschus, "Towards a next-generation matrix library for Java," *COMPSAC: International Computer Software and Applications Conference*, 2009, in press.