Journal of Machine Learning Research 10 (2009) 931-934 Submitted 9/08; Revised 1/09; Published 4/09

Java-ML: A Machine Learning Library

Thomas Abeel THOMAS.ABEEL@PSB.UGENT.BE
Yves Van de Peer YVES.VANDEPEER@PSB.UGENT.BE
Yvan Saeys YVAN.SAEYS@PSB.UGENT.BE
VIB Department of Plant Systems Biology
Ghent University
9000 Gent, Belgium

Editor: Sören Sonnenburg

Abstract

Java-ML is a collection of machine learning and data mining algorithms, which aims to be a readily usable and easily extensible API for both software developers and research scientists. The interfaces for each type of algorithm are kept simple and algorithms strictly follow their respective interface. Comparing different classifiers or clustering algorithms is therefore straightforward, and implementing new algorithms is also easy. The implementations of the algorithms are clearly written, properly documented and can thus be used as a reference. The library is written in Java and is available from http://java-ml.sourceforge.net/ under the GNU GPL license.

Keywords: open source, machine learning, data mining, java library, clustering, feature selection, classification

1. Introduction

Machine learning techniques are increasingly popular in research fields like bio- and chemo-informatics, text and web mining, as well as many other areas of research and industry. In this paper we present Java-ML: a cross-platform, open source machine learning library written in Java.

Several well-known data mining libraries already exist, including for example, Weka (Witten and Frank, 2005) and Yale/RapidMiner (Mierswa et al., 2006). These programs provide a user-friendly interface and are geared towards interactive use with the user. In contrast to these programs, Java-ML is oriented towards developers that want to use machine learning in their own programs. To this end, Java-ML interfaces are restricted to the essentials, and are very easy to understand. As a result, Java-ML facilitates a broad exploration of different models, is straightforward to integrate into your own source code, and can be easily extended.

Regarding the content of the library, Java-ML also has a different focus than the other libraries. Java-ML contains an extensive set of similarity based techniques, and offers state-of-the-art feature selection techniques. The large number of similarity functions allow for a broad set of clustering and instance based learning techniques, while the feature selection techniques are well suited to deal with high-dimensional domains, such as the ones often encountered in bioinformatics and biomedical applications.

Clustering
K-means-like (7)
Self organizing maps
Density based clustering (3)
Markov chain clustering
Cobweb
Cluster evaluation measures (15)

Classification
SVM (2)
Instance based learning (4)
Tree based methods (2)
Random Forests
Bagging

Feature selection
Entropy based methods (4)
Stepwise addition/removal (2)
SVM_RFE
Random forests
Ensemble feature selection

Data filters
Discretization
Normalization (2)
Missing values (3)
Instance manipulation (11)

Distance measures
Similarity measures (6)
Distance metrics (11)
Correlation measures (2)

Utilities
Cross-validation/evaluation
Data loading (ARFF and CSV)
Weka bridges (2)

Table 1: Overview of the main algorithms included in Java-ML. The number of algorithms for each category is shown in parentheses.

2. Description of the Library

In this section we first describe the software design of Java-ML, we then discuss how to integrate it in your program and finally we cover the documentation.

2.1 Structure of the Library

The library is built around two core interfaces: `Dataset` and `Instance`. These two interfaces have several implementations for different types of samples. The machine learning algorithms implement one of the following interfaces: `Clusterer`, `Classifier`, `FeatureScoring`, `FeatureRanking` or `FeatureSubsetSelection`. Distance, correlation and similarity measures implement the interface `DistanceMeasure`. These distance measures can be used in many algorithms to modify their behavior. Cluster evaluation measures are defined by the `ClusterEvaluation` interface. Manipulation filters either implement `InstanceFilter` or `DatasetFilter`, depending on the level they work on. All implementing classes for each of the interfaces are available from the API documentation that is available on the Java-ML website. Each of these interfaces provides one or two methods that are required to execute the algorithm on a particular data set. Several utility classes make it easy to load data from tab or comma separated files and from ARFF formatted files. An overview of the main algorithms included in Java-ML can be found in Table 1.

The library provides several algorithms that have not been made available before in a bundled form. In particular, clustering algorithms and the accompanying cluster evaluation measures are extensively represented. This includes the adaptive quality-based clustering algorithm, density based methods, self-organizing maps (both as clustering and classification algorithm) and numerous other

well-known clustering algorithms. A large number of distance, similarity and correlation measures are included. Feature selection algorithms include traditional algorithms like symmetrical uncertainty, gain ratio, RELIEF, stepwise addition/removal, as well as a number of more recent methods (SVMRFE and random forest attribute evaluation). Also the recently introduced concept of ensemble feature selection techniques (Saeys et al., 2008) is incorporated in the library. We have also implemented a fast and simple random tree algorithm to cope with high dimensional, sparse and ambiguous data. Finally, we provide bridges for classification and clustering in Weka and libsvm (Fan et al., 2005).

2.2 Easy Integration in Your Own Source Code

Including Java-ML algorithms in your own source code is very simple. To illustrate this, we present here two short code fragments that demonstrate the ease to integrate the library. The following lines of code integrate a K-Means clustering algorithm in your own program.

```
Dataset data = FileHandler.loadDataset(new File("iris.data"), 4, ",");
Clusterer km = new KMeans();
Dataset[]clusters=km.cluster(data);
```

The first line uses the FileHandler utility to load data from the iris.data file. In this file, the class label is on the fourth position and the fields are separated by a comma. The second line constructs a new instance of the KMeans clustering algorithm with default values, in this case k=4. The third line uses the KMeans instance to cluster the data that we loaded in the first line. The resulting clusters will be returned as an array of data sets.

The following example illustrates how to perform a cross-validation experiment for a specific dataset and classifier.

```
Dataset data = FileHandler.loadDataset(new File("iris.data"), 4, ",");
Classifier knn = new KNearestNeighbors(5);
CrossValidation cv = new CrossValidation(knn);
Map<Object, PerformanceMeasure> p = cv.crossValidation(data);
```

First we load the iris data set, and construct a K-nearest neighbors classifier, which uses 5 neighbors to classify instances. In the next line, we initialize the cross-validation with our classifier. The last line runs the cross-validation on the loaded data. By default, a 10-fold cross validation will be performed. The result is returned in a map, which maps each class label to its corresponding PerformanceMeasure (`Map<Object,PerformanceMeasure>`). For classification problems, a performance measure is a wrapper around four values: (i) true positives, (ii) true negatives, (iii) false positives and (iv) false negatives. This class also provides a number of derivative measures such as accurracy, error rate, precision, recall and others. More advanced samples are available from the documentation pages on the Java-ML website.

2.3 Documentation

There are a number of resources for documentation about Java-ML. The source code itself is documented thoroughly, always up-to-date, and accessible from the web site through the API documentation. The web site additionally provides a number of tutorials with illustrated code samples for

the most common tasks in Java-ML, covering the following topics: installing the library, introducing basic concepts, creating and loading data, creating algorithms and applying them to your data, and more advanced topics for people who would like to contribute to the library. Finally, all code samples as well as the PDF versions of the tutorials are also included in the Java-ML distribution itself.

3. Case Studies

The library described in this manuscript has been used in several studies. Here we highlight two applications which have been recently published.

Initially, the project focused on clustering algorithms and measures to evaluate the quality of a clustering. Our goal was to separate DNA sequences that are likely to contain a promoter (the controlling element of a gene) from other sequences, a well-known task in bioinformatics. The best results were obtained using a clustering algorithm based on self-organizing maps (Abeel et al., 2008).

More recently, the focus has shifted toward feature selection. More specifically, we are looking whether ensemble feature selection (combining different feature selectors) can improve the stability of feature selection in case of high-dimensional data sets with few samples. The improvements in stability were shown not to affect the prediction accuracy. This is ongoing research, but the first results are promising (Saeys et al., 2008).

Acknowledgments

We thank A. De Rijcke for his early contributions to Java-ML, as well as the anonymous reviewers for their valuable comments. TA is funded by IWT-Vlaanderen. YS would like to thank the Research Foundation-Flanders (FWO-Vlaanderen) for funding his research.

References

Thomas Abeel, Yvan Saeys, Pierre Rouzé, and Yves Van de Peer. ProSOM: Core promoter prediction based on unsupervised clustering of DNA physical profiles. *Bioinformatics*, 24(13):i24–i31, July 2008.

Rong-En Fan, Pai-Hsuen Chen, and Chih-Jen Lin. Working set selection using the second order information for training SVM. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. Yale: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-06)*, 2006.

Yvan Saeys, Thomas Abeel, and Yves Van de Peer. Robust feature selection using ensemble feature selection techniques. In *Proceedings of the ECML-PKDD conference 2008*, 2008.

Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.