Journal of Machine Learning Research 13 (2012) 555-559 Submitted 6/11; Revised 2/12; Published 3/12

# ML-Flex: A Flexible Toolbox for Performing Classification Analyses In Parallel

**Stephen R. Piccolo** STEPHEN.PICCOLO@HSC.UTAH.EDU
*Department of Pharmacology and Toxicology, School of Pharmacy*
*University of Utah*
*Salt Lake City, UT 84112, USA*

**Lewis J. Frey** LEWIS.FREY@HSC.UTAH.EDU
*Huntsman Cancer Institute*
*Department of Biomedical Informatics, School of Medicine*
*University of Utah*
*Salt Lake City, UT 84112, USA*

**Editor:** Geoff Holmes

## Abstract

Motivated by a need to classify high-dimensional, heterogeneous data from the bioinformatics domain, we developed ML-Flex, a machine-learning toolbox that enables users to perform two-class and multi-class classification analyses in a systematic yet flexible manner. ML-Flex was written in Java but is capable of interfacing with third-party packages written in other programming languages. It can handle multiple input-data formats and supports a variety of customizations. ML-Flex provides implementations of various validation strategies, which can be executed in parallel across multiple computing cores, processors, and nodes. Additionally, ML-Flex supports aggregating evidence across multiple algorithms and data sets via ensemble learning. This open-source software package is freely available from `http://mlflex.sourceforge.net`.

**Keywords:** toolbox, classification, parallel, ensemble, reproducible research

## 1. Introduction

The machine-learning community has developed a wide array of classification algorithms, but they are implemented in diverse programming languages, have heterogeneous interfaces, and require disparate file formats. Also, because input data come in assorted formats, custom transformations often must precede classification. To address these challenges, we developed ML-Flex, a general-purpose toolbox for performing two-class and multi-class classification analyses. Via command-line interfaces, ML-Flex can invoke algorithms implemented in any programming language. Currently, ML-Flex can interface with several popular third-party packages, including *Weka* (Hall et al., 2009), *Orange* (Demsar et al., 2004), *C5.0 Decision Trees* (RuleQuest Research, 2011), and *R* (R Development Core Team, 2011). In many cases, new packages can be integrated with ML-Flex through only minor modifications to configuration files. However, via a simple extension mechanism, ML-Flex also supports a great amount of flexibility for custom integrations. ML-Flex can parse input data in delimited and ARFF formats, and it can easily be extended to parse data from custom sources.

ML-Flex can perform systematic evaluations across multiple algorithms and data sets. Furthermore, it can aggregate evidence across algorithms and data sets via ensemble learning. The

following ensemble learners currently are supported: majority vote (Boland, 1989), weighted majority vote (Littlestone and Warmuth, 1994), mean probability rule, weighted mean probability rule, maximum probability rule, select-best rule, and stacked generalization (Wolpert, 1992). (When ensemble learners are applied, predictions from individual classifiers are reused from prior execution steps, thus decreasing computational overhead.)

ML-Flex provides implementations of various validation strategies, including simple train-test validation, K-fold cross validation, repeated random sampling validation, and leave-one-out cross validation. For each validation strategy, ML-Flex can also apply feature selection/ranking algorithms and perform nested cross-validation within respective training sets. To enable shorter execution times for computationally intensive validation strategies, ML-Flex can be executed in parallel across multiple computing cores/processors and multiple nodes on a network. Individual computing nodes may have heterogeneous hardware configurations so long as each node can access a shared file system. In a recent analysis of a large biomedical data set, ML-Flex was executed simultaneously across hundreds of cluster-computing cores (Piccolo, 2011).

Upon completing classification tasks, ML-Flex produces parsable text files that report performance metrics, confusion matrices, outputs from individual algorithms, and a record of all configuration settings used. A formatted HTML report with the same information is also provided. These features enable reproducibility and transparency about how a given set of results was obtained.

Available from `http://mlflex.sourceforge.net`, ML-Flex is licensed under the GNU General Public License Version 3.0. The distribution contains extensive documentation, including tutorials and sample experiments.

## 2. Architecture

Execution of ML-Flex revolves around the concept of an "experiment." For a given experiment, the user specifies one or more sets of independent (predictor) variables and a dependent variable (class) as well as any algorithm(s) that should be applied to the data. Various other settings (for example, number of cross-validation folds, number of iterations, random seed) can be altered optionally.

To configure an experiment, users can specify three types of settings: 1) learner templates, 2) algorithm parameters, and 3) experiment-specific preferences. For example, if a user wanted to apply the *Weka* implementation of the *One Rule* classification algorithm, with a minimum bucket size of 6, to the classic Iris data set, she would first create a learner template such as the following (simplified for brevity):

```
wekac;mlflex.learners.WekaLearner;java -classpath lib/weka.jar {ALGORITHM}
-t {INPUT_TRAINING_FILE} -T {INPUT_TEST_FILE} -p 0 -distribution
```

The above template contains three items, separated by semicolons: 1) a unique key, 2) the name of a Java class that supports interfacing with Weka, and 3) a templated shell command for invoking Weka on that system. (When ML-Flex executes a command, placeholders—for example, "{ALGORITHM}"—are replaced with relevant values.) Having specified this template, the user would specify the following in an algorithm-parameters file:

```
one_r;wekac;weka.classifiers.rules.OneR -B 6
```

This entry indicates 1) a unique key, 2) a reference to the learner template, and 3) the parameters that should be passed to Weka. Finally, the user would include the following in an experiment file:

```
CLASSIFICATION_ALGORITHMS=one_r
DATA_PROCESSORS=mlflex.dataprocessors.ArffDataProcessor("iris.arff")
```

The first line references the algorithm-parameters entry, and the second line indicates the name of a Java class that can parse the input data. (Example files and detailed explanations of all configuration settings are provided.)

At each stage of an experiment, ML-Flex can execute in parallel, using a simple, coarse-grained architecture. Independent computing tasks—for example, parsing a given input file, classifying a given combination of data set and algorithm and cross-validation fold, or outputting results—are packaged by each computing node into uniquely defined Java objects. Then thread(s) on each computing node compete to execute each task via a locking mechanism. Initially, each thread checks for a status file that would indicate whether a given task has been executed and the corresponding result has been stored. If the task has not yet been processed, the thread checks the file system for a correspondingly named "lock file" that indicates whether the task is currently being processed by another thread. If no lock file exists, the thread attempts to create the file atomically. Having successfully created the lock file, the thread executes the task, stores the result on the file system, and deletes the lock file. If a system error or outage occurs, the lock file will persist for a user-configurable period of time, after which it will be deleted and the task reattempted.

Because this parallel-processing approach requires many input/output operations and because individual computing nodes do not communicate with each other directly, minor inefficiencies may arise. However, the simplicity of the approach offers many advantages: 1) no third-party software package is necessary for interprocess communication, 2) individual computing nodes may run different operating systems and/or have different hardware configurations, 3) the number of computing nodes that can be employed simultaneously is scalable, 4) if an individual computing node goes offline, remaining nodes are unaffected, 5) additional computing nodes may be assigned after a job has already begun processing, and 6) experiments are restartable. The latter three features are particularly desirable in cluster-computing environments where server reliability may be less than optimal and where computing nodes may become available incrementally.

## 3. Related Work

Machine-learning toolboxes like *caret*, *Weka*, *Orange*, and *KNIME* implement a broad range of classification algorithms, but many useful algorithms are not included in these packages, perhaps due to licensing restrictions, resource constraints, or programming-language preferences. Like *SHOGUN* (Sonnenburg et al., 2010), ML-Flex provides a harness that allows developers to implement algorithms natively in the language of their choice. Because no language-specific interfaces are necessary in ML-Flex, integration can often occur with no change to the ML-Flex source code. This approach also empowers algorithm developers to take the lead in integrating with ML-Flex and thus benefit from its other features, including model evaluation and parallelization.

*KNIME* and *RapidMiner* (Mierswa et al., 2006) support various input-file formats, transformation procedures, and data-filtering modules. In an alternative approach, ML-Flex provides an extension mechanism, which allows users to preprocess data using custom Java code. (*Weka* supports similar functionality.) This approach may be especially useful in research settings where unusual data formats are prevalent, advanced transformations are desired, or data must be accessed remotely (for example, via Web services, including those that require authentication).

Other toolboxes support the ability to distribute workloads across multiple computers. For example, a client machine executing the *Weka* Experimenter module can distribute its workload via Java Remote Method Invocation. The *caret* R package (Kuhn, 2008) uses the *NetWorkSpaces*™ technology to distribute workloads. The commercial version of *KNIME* (Berthold et al., 2007) can distribute its workload to cluster servers running *Oracle® Grid Engine*. And *Apache Mahout*™ (Ingersol, 2009) uses the map/reduce paradigm to enable execution on cluster-computing environments. ML-Flex differentiates itself from these tools by 1) supporting heterogeneous configurations among computing nodes, 2) allowing recovery from system outages due to no single point of failure (assuming redundant disk storage), and 3) supporting restartability and incremental node allocations.

Many toolboxes—including *Weka*, *Orange*, *KNIME*, *caret*, *SHOGUN*, and *Waffles* (Gashler, 2011)—support experimental reproducibility via application programming interfaces (API), command-line interfaces (CLI), and/or visual workflow pipelines. Users can write client tools that invoke APIs and that can later be re-executed. Scripts that invoke CLIs can be repeated; and visual pipelines typically encapsulate execution logic. In ML-Flex, users encode all configuration settings in text files. With this approach, users are not required to write code nor extensive scripts. However, with a modest scripting effort, it is possible to generate configuration files dynamically, an approach that may not be feasible with visual workflows. Because a copy of relevant configuration files accompany the results of each experiment, subsequent replication of results is straightforward. Additionally, in experiments that use repeated random sampling validation, ML-Flex encapsulates sampling and summarization logic, which may be burdensome to replicate with alternative approaches.

## Acknowledgments

While developing this software, SRP was funded by a U.S. National Library of Medicine training fellowship (1T15-LM007124). We gratefully acknowledge an allocation of computer time from the Fulton Supercomputing Lab at Brigham Young University. We also thank the reviewers for thoughtful, detailed feedback.

## References

M. Berthold, N. Cebron, F. Dill, T. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel. KNIME: The Konstanz information miner. In *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)*, Springer, 2007.

P. Boland. Majority systems and the condorcet jury theorem. *The Statistician*, 38(3):181–189, 1989.

J. Demsar, B. Zupan, G. Leban, and T. Curk. Orange: From experimental machine learning to interactive data mining. In *Knowledge Discovery in Databases: PKDD 2004*, pages 537–539, Berlin, 2004.

M. Gashler. Waffles: A machine learning toolkit. *Journal of Machine Learning Research*, 12(Jul):2383–2387, 2011.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The WEKA data mining software. *ACM SIGKDD Explorations Newsletter*, 11(1):10, 2009.

G. Ingersol. Introducing Apache Mahout. *IBM developerWorks Technical Library*, 2009. Available electronically at http://www.ibm.com/developerworks/java/library/j-mahout/.

M. Kuhn. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5), 2008.

N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. YALE: Rapid prototyping for complex data mining tasks. In *KDD '06: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (August 2006)*, ACM, 2006.

S. Piccolo. *Informatics Framework For Evaluating Multivariate Prognosis Models: Application to Human Glioblastoma Multiforme*. PhD dissertation, University of Utah, Salt Lake City, Utah, 2011.

R Development Core Team. *R: A Language and Environment For Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. Available electronically at http://www.R-project.org.

Rulequest Research. Data mining tools See5 and C5.0. Available electronically at http://www.rulequest.com/see5-info.html.

S. Sonnenburg, G. Ratsch, S. Henschel, C. Widmer, J. Behr, A. Zien, F. Bona, A. Binder, C. Gehl and V. Franc. The SHOGUN machine learning toolbox. *Journal of Machine Learning Research*, 11(Jun):1799–1802, 2010.

D. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.