Journal of Machine Learning Research 16 (2015) 1243-1247 Submitted 11/14; Published 6/15

Encog: Library of Interchangeable Machine Learning Models for Java and C#

Jeff Heaton JEFFHEATON@ACM.ORG
College of Engineering and Computing
Nova Southeastern University
Fort Lauderdale, FL 33314, USA

Editor: Cheng Soon Ong

Abstract

This paper introduces the Encog library for Java and C#, a scalable, adaptable, multi-platform machine learning framework that was first released in 2008. Encog allows a variety of machine learning models to be applied to data sets using regression, classification, and clustering. Various supported machine learning models can be used interchangeably with minimal recoding. Encog uses efficient multithreaded code to reduce training time by exploiting modern multicore processors. The current version of Encog can be downloaded from `http://www.encog.org`.

Keywords: Java, C#, neural network, support vector machine, open source software

1. Intention and Goals

This paper describes the Encog API for Java and C# that is provided as a JAR or DLL library. The C# version of Encog is also compatible with the Xamarin Mono package. Encog has an active community that has provided many enhancements that are beyond the scope of this paper. This includes extensions such as Javascript, GPU processing, C/C++ support, Scala support, and interfaces to various automated trading platforms. The scope of this paper is limited to the Java and C# API.

Encog allows the Java or C# programmer to experiment with a wide range of machine language models using a simple, consistent interface for clustering, regression, and classifications. This allows the programmer to construct applications that discover which model provides the most suitable fit for the data. Encog provides basic tools for automated model selection. Most Encog models are implemented as efficient multithreaded algorithms to reduce processing time. This often allows Encog to perform more efficiently than many other Java and C# libraries, as demonstrated empirically by Taheri (2014) and Matviykiv and Faitas (2012). Luhasz et al. (2013) and Ramos-Pollán et al. (2012) also saw favorable results when evaluating Encog to similar libraries.

The Encog's API is presented in an intuitive object-oriented paradigm that allows various models, optimization algorithms, and training algorithms to be highly interchangeable. However, beneath the API, the models are represented as one and two-dimensional arrays. This internal representation allows for highly efficient calculation. The API shields the programmer from the complexity of model calculation and fitting.

Encog contains nearly 400 unit tests to ensure consistency between the Java and C# model implementations. Expected results are calculated and cross-checked between the two platforms. A custom pseudorandom number generator (PRNG) is used in both language's unit tests to ensure that even stochastic models produce consistent, verifiable test results.

Encog contains nearly 150 examples to demonstrate the use of the API in a variety of scenarios. These examples include simple prediction, time series, simulation, financial applications, path finding, curve fitting, and other applications. Documentation for Encog is provided as Java/C# docs and an online wiki. Additionally, discussion groups and a Stack Overflow tag are maintained for support. Links to all of these resources can be found at `http://www.encog.org`.

2. Framework Overview

The design goal of Encog is to provide interchangeable models with efficient, internal implementations. The Encog framework supports machine learning models with multiple training algorithms. These models are listed here:

- Adaline, Feedforward, Hopfield, PNN/GRNN, RBF & NEAT neural networks
- generalized linear regression (GLM)
- genetic programming (tree-based)
- k-means clustering
- k-nearest neighbors
- linear regression
- self-organizing map (SOM)
- simple recurrent network (Elman and Jordan)
- support vector machine (SVM)

Encog provides optimization algorithms such as particle swarm optimization (PSO) (Poli, 2008), genetic algorithms (GA), Nelder-Mead and simulated annealing. These algorithms can optimize a vector to minimize a loss function; consequently, these algorithms can fit model parameters to data sets.

Propagation-training algorithms for neural network fitting, such as back propagation (Rumelhart et al., 1988), resilient propagation (Riedmiller and Braun, 1992), Levenberg-Marquardt (Marquardt, 1963), quickpropagation (Fahlman, 1988), and scaled conjugate gradient (Møller, 1993) are included. Neural network pruning and model selection can be used to find optimal network architectures. Neural network architectures can be automatically built by a genetic algorithm using NEAT and HyperNEAT (Stanley and Miikkulainen, 2002).

A number of preprocessing tools are built into the Encog library. Collected data can be divided into training, test, and validation sets. Time-series data can be encoded into data windows. Quantitative data can be normalized by range or z-score to prevent biases in some models. Masters (1993) normalizes qualitative data using one-of-n encoding or equilateral encoding. Encog uses these normalization techniques.

Encog also contains extensive support for genetic programming using a tree representation (Koza, 1993). A full set of mathematical and programming functions are provided. Additionally, new functions can be defined. Constant nodes can either be drawn from a constant pool or generated as needed. Rules can optionally be added to simplify expressions and penalize specific genome patterns.

3. API Overview

One of the central design philosophies of Encog is to allow models to be quickly interchanged without a great deal of code modification. A classification example will demonstrate this interchangeability, using the iris data set (Fisher, 1936). Portions of this classification example are presented in this paper, using the Java programming language. The complete example, in both Java and C#, is provided in the *Encog Quick Start Guide* (available from `http://www.encog.org`). The Quick Start Guide also provides regression and time-series examples.

The following example learns to predict the species of an iris flower by using four types of measurements from each flower. To begin, the program loads the iris data set's CSV file. In addition to CSV, Encog contains classes to read fixed-length text, JDBC, ODBC, and XML data sources. The iris data set is loaded, and the four measurement columns are defined as continuous values.

```
VersatileDataSource source = new CSVDataSource(irisFile, false,
    CSVFormat.DECIMAL_POINT);
VersatileMLDataSet data = new VersatileMLDataSet(source);
data.defineSourceColumn("sepal-length", 0, ColumnType.continuous);
data.defineSourceColumn("sepal-width", 1, ColumnType.continuous);
data.defineSourceColumn("petal-length", 2, ColumnType.continuous);
data.defineSourceColumn("petal-width", 3, ColumnType.continuous);
```

The species of Iris is defined as nominal value. Defining the columns as continuous, nominal or ordinal allows Encog to determine the appropriate way to encode these data for a model. For specialized cases, it is possible to override Encog's encoding defaults for any model type.

```
ColumnDefinition outputColumn = data.defineSourceColumn("species", 4,
    ColumnType.nominal);
```

Once the columns have been defined, the file is analyzed to determine minimum, maximum, and other statistical properties of the columns. This allows the columns to be properly normalized and encoded by Encog for modeling.

```
data.analyze();
data.defineSingleOutputOthersInput(outputColumn);
```

Next the model type is defined to be a feedforward neural network.

```
EncogModel model = new EncogModel(data);
model.selectMethod(data, MLMethodFactory.TYPE_FEEDFORWARD);
```

Only the above line needs to be changed to switch to model types that include the following:

- **MLMethodFactory.SVM**: support vector machine
- **MLMethodFactory.TYPE_RBFNETWORK**: RBF neural network

- **MLMethodFactor.TYPE_NEAT**: NEAT neural network
- **MLMethodFactor.TYPE_PNN**: probabilistic neural network

Next the data set is normalized and encoded. Encog will automatically determine the correct normalization type based on the model chosen in the last step. For model validation, 30% of the data are held back. Though the validation sampling is random, a seed of 1001 is used so that the items selected for validation remain constant between program runs. Finally, the default training type is selected.

```
data.normalize();
model.holdBackValidation(0.3, true, 1001);
model.selectTrainingType(data);
```

The example trains using a 5-fold cross-validated technique that chooses the model with the best validation score. The resulting training and validation errors are displayed.

```
MLRegression bestMethod = (MLRegression)model.crossvalidate(5, true);
System.out.println( "Training error: " + EncogUtility.calculateRegressionError(
    bestMethod, model.getTrainingDataset()));
System.out.println( "Validation error: " + EncogUtility.calculateRegressionError
    (bestMethod, model.getValidationDataset()));
```

Display normalization parameters and final model.

```
NormalizationHelper helper = data.getNormHelper();
System.out.println(helper.toString());
System.out.println("Final model: " + bestMethod);
```

4. Future Plans and Conclusions

A number of enhancements are planned for Encog. Gradient boosting machines (GBM) and deep learning are two future model additions. Several planned enhancements will provide interoperability with other machine learning packages. Future versions of Encog will have the ability to read and write Weka Attribute-Relation File Format (ARFF) and libsvm data files. Encog will gain the ability to load and save models in the Predictive Model Markup Language (PMML) format. A code contribution by Mosca (2012) will soon be integrated, enhancing Encog's ensemble learning capabilities.

Acknowledgments

The Encog community has been very helpful for bug reports, bug fixes, and feature suggestions. Contributors to Encog include Olivier Guiglionda, Seema Singh, César Roberto de Souza, and others. A complete list of contributors to Encog can be found at the GitHub repository: `https://github.com/encog`. Alan Mosca, Department of Computer Science and Information Systems, Birkbeck, University of London, UK, created Encog's ensemble functionality. Matthew Dean, Marc Fletcher and Edmund Owen, Semiconductor Physics Research Group, University of Cambridge, UK, created Encog's RBF Neural network model.

References

S. Fahlman. An empirical study of learning speed in back-propagation networks. Technical report, Carnegie Mellon University, 1988.

R. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(7):179–188, 1936.

J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Complex adaptive systems. MIT Press, 1993. ISBN 978-0-262-11170-6.

G. Luhasz, V. Munteanu, and V. Negru. Data mining considerations for knowledge acquisition in real time strategy games. In *IEEE 11th International Symposium on Intelligent Systems and Informatics, SISY 2013, Subotica, Serbia, September 26-28, 2013*, pages 331–336, 2013. doi: 10.1109/SISY.2013.6662596.

D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11(2):431–441, 1963. doi: 10.1137/0111030.

T. Masters. *Practical Neural Network Recipes in C++*. Academic Press Professional, Inc., San Diego, CA, USA, 1993. ISBN 0-12-479040-2.

O. Matviykiv and O. Faitas. Data classification of spectrum analysis using neural network. *Lviv Polytechnic National University*, 2012.

M. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533, 1993.

A. Mosca. Extending Encog: a study on classifier ensemble techniques. Master's thesis, Birkbeck, University of London, 2012.

R. Poli. Analysis of the publications on the applications of particle swarm optimisation. *J. Artif. Evol. App.*, 2008:4:1–4:10, January 2008. ISSN 1687-6229.

Raúl Ramos-Pollán, Miguel Ángel Guevara-López, and Eugénio C. Oliveira. A software framework for building biomedical machine learning classifiers through grid computing resources. *J. Medical Systems*, 36(4):2245–2257, 2012. doi: 10.1007/s10916-011-9692-3.

M. Riedmiller and H. Braun. Rprop: A fast adaptive learning algorithm. Technical report, Proc. of ISCIS VII), Universitat, 1992.

D. Rumelhart, G. Hinton, and R. Williams. Neurocomputing: Foundations of research. In James A. Anderson and Edward Rosenfeld, editors, *Neurocomputing: Foundations of Research*, chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01097-6.

K. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, June 2002. ISSN 1063-6560.

T. Taheri. Benchmarking and comparing Encog, Neuroph and JOONE neural networks. `http://goo.gl/A56iyx`, June 2014. Accessed: 2014-10-9.