Journal of Machine Learning Research 10 (2009) 743-746 Submitted 11/07; Revised 7/08; Published 3/09

Nieme: Large-Scale Energy-Based Models

Francis Maes FRANCIS.MAES@LIP6.FR
Universite Pierre et Marie Curie
Laboratoire d'Informatique de Paris 6, UMR CNRS 7606
104, Avenue du President Kennedy
Paris, France

Editor: Cheng Soon Ong

Abstract

In this paper we introduce NIEME,[1] a machine learning library for large-scale classification, regression and ranking. NIEME relies on the framework of *energy-based models* (LeCun et al., 2006) which unifies several learning algorithms ranging from simple perceptrons to recent models such as the pegasos support vector machine or l1-regularized maximum entropy models. This framework also unifies batch and stochastic learning which are both seen as energy minimization problems. NIEME can hence be used in a wide range of situations, but is particularly interesting for large-scale learning tasks where both the examples and the features are processed incrementally. Being able to deal with new incoming features at any time within the learning process is another original feature of the NIEME toolbox. NIEME is released under the GPL license. It is efficiently implemented in C++, it works on Linux, Mac OS X and Windows and provides interfaces for C++, Java and Python.

Keywords: large-scale machine learning, classification, ranking, regression, energy-based models, machine learning software

1. Introduction

Although many machine learning toolkits have been developed in the past years, it is often the case that they do not scale well to real-world applications. Developing learning algorithms for large-scale applications is a challenging design and implementation problem. In this paper, we introduce NIEME, a C++ library that provides generic tools for large-scale learning. NIEME covers two domains: supervised learning and learning in decision processes. In this paper, we focus on the supervised learning part, made of classification, regression and ranking algorithms. The support of decision processes is still in active development and will be more detailed in further versions of the library.

2. Framework: Energy-based Models

Most learning machines in NIEME rely on the unified framework of energy-based models introduced by LeCun et al. (2006). In this framework, illustrated in Figure 1, a learning machine can be interpreted as a combination of an architecture $\mathcal{A}$ with parameters θ, a per-example loss L , a set of regularizers $\{\Omega_1, \ldots, \Omega_R\}$ and a learner $\mathcal{L}$. Given the architecture, the per-example loss and the

1. Code for NIEME can be found at `http://nieme.lip6.fr`.

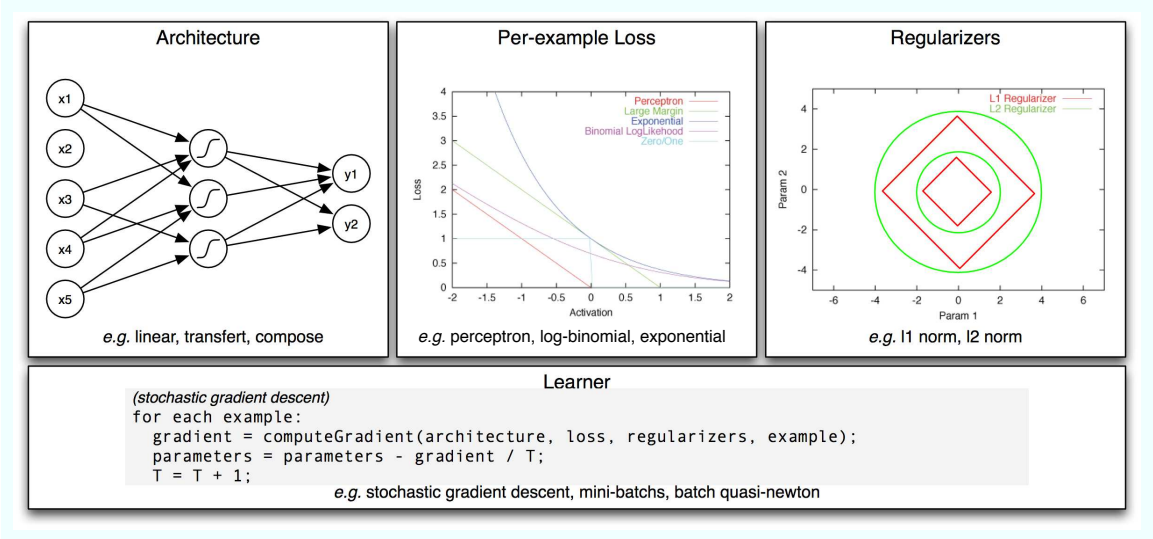

Figure 1: This figure illustrates the framework of energy-based models. Top: the three components that define the learning energy. Bottom: the *learner* component that performs energy minimization.

Model	Architecture	Loss	Regularizers	Learner
Perceptron	linear	perceptron	none	stochastic descent
Logistic regression	linear	log-binomial	none	batch quasi-newton
Pegasos linear SVM	linear	hinge loss	l2	pegasos learner
Multilayer perceptron	linear ∘ transfer ∘ linear	perceptron	none	stochastic descent
L1-maxent classifier	multi-class linear	log-binomial	l1	batch quasi-newton
Pegasos multi-class SVM	multi-class linear	hinge loss	l2	pegasos learner
Least-square regression	linear	squared loss	none	batch quasi-newton
Custom	linear ∘ transfer	absolute loss	l1 + l2	batch rprop
Many others	...	...	...	...

Table 1: This table gives some examples of energy-based models. Each model is defined by its architecture, per-example loss, regularizers and learner. The ∘ symbols denotes architecture composition.

regularizers, we can define a learning loss for a set of examples $S = \{e_1, \ldots e_N\}$:

$$\mathcal{L}_S(\theta) = \frac{1}{N}\sum_{i=1}^{N} L(e_i, \mathcal{A}_\theta) + \sum_{i=1}^{R} \Omega_i(\theta).$$

The aim of the learner is then to find parameters θ that minimize the learning loss given the training set S. As illustrated in Table 1, many combinations are possible in the energy-based framework. Some correspond to well-known learning machines, others to more original approaches. We describe the three components that define the learning loss $\mathcal{L}_S(\theta)$ in Section 2.1, describe learners in Section 2.2 and discuss some aspects of features in NIEME in Section 2.3.

2.1 Learning Loss components

The architecture is a parameterized function which computes predictions given input vectors. A simple example is the linear architecture, which computes a single output as a scalar product be-

tween the input vector and the parameter vector. NIEME supports elementary architectures (linear, multi-class linear, neural network transfer function) as well as a *composition* operation which allows users to create a new architecture by chaining existing ones. The per-example loss quantifies *how bad* an architecture and its parameters perform on a given learning example. Learning aims at finding parameters which lead to low expected per-example loss. Currently NIEME implements four different loss functions for discriminant learning (perceptron loss, hinge loss, log-binomial loss and exponential loss) and two loss functions for regression (squared and absolute loss). Regularizers are functions which measure the *complexity* of an architecture and its parameters. It has been shown (Vapnik, 1999) that penalizing complex models often leads to better generalization performance. Up to now, NIEME includes the two most commonly used regularizers: the l1-norm and l2-norm of the parameters.

2.2 Learners

The learner is the algorithmic component that performs learning loss minimization. NIEME implements three batch learners: the large-scale limited-memory quasi-Newton method of Lio and Nocedal (1989), the recently proposed method of Andrew and Gao (2007) for minimizing large-scale l1-regularized models and the rprop method of Riedmiller and Braun (1993). If batch learning is not possible for a given problem, NIEME proposes classical online methods such as stochastic gradient descent. Finally, NIEME also offers mini-batch methods including the recent SVM solver of Shalev-Shwartz et al. (2007).

2.3 Feature Space

In large-scale applications, such as text processing, natural language processing, or bioinformatics, examples are often described with sparse feature representations. NIEME has thus been designed for efficient processing of such vectors. Moreover, in online settings, NIEME has the key ability to handle new incoming features at any time within the learning process. Everytime a new feature appears, the parameters of the learning machines are automatically extended to include this new feature. This is particularly interesting when dealing with large data streams for which the feature set cannot be known entirely before learning. Conceptually, all those features exist from the beginning of the learning process although most of them have zero values. Practically, a feature does not affect the learning parameters until it has been seen once.

3. Implementation

The core of NIEME is implemented in portable C++ and compiles currently under Windows (with Visual C++), Mac OS X (with Xcode or Makefiles) and Linux (with Makefiles or KDevelop). The implementation of about 14,000 lines of code is fully object-oriented and makes use of several design-patterns. The code is clear and easy to extend. However, it is not necessary to have a detailed understanding of the implementation in order to use NIEME. Indeed, NIEME includes an easy-to-use interface that can be used from C++, Java or Python, as illustrated in Figure 2. Wrappers for Java and Python are automatically generated thanks to the SWIG tool[2] that makes the glue code. NIEME could even be extended to support languages such as C# or OCaml.

2. SWIG can be found at `http://www.swig.org`.

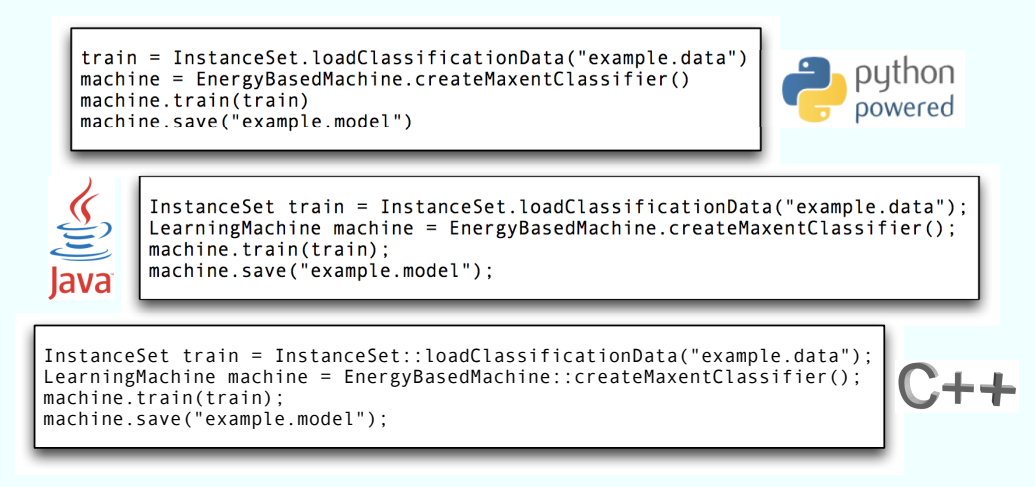

Figure 2: This figure shows the same program in three languages (Python, Java and C++). The program loads a classification data set, trains a maximum entropy classifier and saves the resulting model in a file called "example.model".

3.1 Tutorials, Documentation, License, Unit Tests

The NIEME website includes a quick-start guide with compilation instructions and tutorials to get started with NIEME in C++, Python or Java. Furthermore, it includes a complete reference documentation of the interface. NIEME is released under the GPL license. All functions of NIEME's interface are unit-tested within the *python unittest* framework.

References

G. Andrew and J. Gao. Scalable training of L1-regularized log-linear models. In Zoubin Ghahramani, editor, *ICML 2007*, pages 33–40. Omnipress, 2007.

Yann LeCun, Sumit Chopra, Raia Hadsell, Ranzato Marc'Aurelio, and Fu-Jie Huang. A tutorial on energy-based learning. In *Predicting Structured Data*. MIT Press, 2006.

D. C. Lio and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Programming*, 45(3):503–528, 1989.

Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *ICNN*, pages 586–591, San Francisco, CA, 1993.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: primal estimated sub-gradient solver for SVM. In Zoubin Ghahramani, editor, *ICML 2007*, pages 807–814. Omnipress, 2007.

V. N. Vapnik. An overview of statistical learning theory. *Neural Networks, IEEE Transactions on*, 10(5):988–999, 1999.