SOL: A Library for Scalable Online Learning Algorithms

Yue Wu†,‡, Steven C.H. Hoi†, Chenghao Liu†, Jing Lu†, Doyen Sahoo†, Nenghai Yu‡
†School of Information Systems, Singapore Management University, Singapore
‡Department of EEIS, University of Science and Technology of China, China
chhoi@smu.edu.sg, {wye,ynh}@mail.ustc.edu.cn, {jing.lu.2014,doyen.2014}@phdis.smu.edu.sg

Abstract

SOL is an open-source library for scalable online learning algorithms, and is particularly suitable for learning with high-dimensional data. The library provides a family of regular and sparse online learning algorithms for large-scale binary and multi-class classification tasks with high efficiency, scalability, portability, and extensibility. SOL was implemented in C++, and provided with a collection of easy-to-use command-line tools, python wrappers and library calls for users and developers, as well as comprehensive documents for both beginners and advanced users. SOL is not only a practical machine learning toolbox, but also a comprehensive experimental platform for online learning research. Experiments demonstrate that SOL is highly efficient and scalable for large-scale machine learning with high-dimensional data.

Keywords: Online Learning, Scalable Machine Learning, High Dimensionality, Sparse Learning

1. Introduction

In many big data applications, data is large not only in sample size, but also in feature/dimension size, e.g., web-scale text classification with millions of dimensions. Traditional batch learning algorithms fall short in low efficiency and poor scalability, e.g., high memory consumption and expensive re-training cost for new training data. Online learning represents a family of efficient and scalable algorithms that sequentially learn one example at a time. Some existing toolbox, e.g., LIBOL [1], allows researchers in academia to benchmark different online learning algorithms, but it was not designed for practical developers to tackle online learning with large-scale high-dimensional data in industry.

In this work, we develop SOL as an easy-to-use scalable online learning toolbox for large-scale binary and multi-class classification tasks. It includes a family of ordinary and sparse online learning algorithms, and is highly efficient and scalable for processing high-dimensional data by using (i) parallel threads for both loading and learning the data, and (ii) specially designed data structure for high-dimensional data. The library is implemented in standard C++ with the cross platform ability and there is no dependency on other libraries. To facilitate developing new algorithms, the library is carefully designed and documented with high extensibility. We also provide python wrappers to facilitate experiments and library calls for advanced users. The SOL website is host at `http://SOL.stevenhoi.org` and the software is made available `https://github.com/LIBOL/SOL`.

2. Scalable Online Learning for Large-Scale Linear Classification

2.1. Overview

Online learning operates sequentially to process one example at a time. Consider $\{(\mathbf{x}_t, y_t)|t \in [1,T]\}$ be a sequence of training data examples, where $\mathbf{x}_t \in R^d$ is a d-dimensional vector, $y_t \in \{+1,-1\}$ for binary classification or $y_t \in \{0,\ldots,C-1\}$ for multi-class classification (C classes). As Algorithm 1 shows, at each time step t, the learner receives an incoming example $\mathbf{x}_t$ and then predicts its class label $\hat{y}_t$. Afterward, the true label y_t is revealed and the learner suffers a loss $l_t(y_t,\hat{y}_t)$, e.g., the hinge loss is commonly used $l_t(y_t,\hat{y}_t) = max(0, 1 - y_t \cdot \hat{y}_t)$ for binary classification. For sparse online learning, one can modify the loss with $L1$ regularization $l_t(y_t,\hat{y}_t) = \hat{l}_t(y_t,\hat{y}_t) + \lambda\|\mathbf{w}_t\|_1$ to induce sparsity for the learned model $\mathbf{w}$. At the end of each learning step, the learner decides when and how to update the model.

Algorithm 1: SOL: Online Learning Framework for Linear Classification

Initialize: $\mathbf{w}_1 = 0$;
for t *in* $\{1,\ldots, T\}$ **do**
  Receive $\mathbf{x}_t \in R^d$, predict $\hat{y}_t$, receive true label y_t;
  Suffer loss $l_t(y_t,\hat{y}_t)$;
  if $l_t(y_t,\hat{y}_t)$ **then**
    $\mathbf{w}_{t+1} \leftarrow update(\mathbf{w}_t)$;
  end
end

The goal of our work is to implement most state-of-the-art online learning algorithms to facilitate research and application purposes on the real world large-scale high dimensional data. Especially, we include sparse online learning algorithms which can effectively learn important features from the high dimensional real world data [2]. We provide algorithms for both binary and multi-class problems. These algorithms can also be classified into first order algorithms [3] and second order algorithms [4] from the model's perspective. The implemented algorithms are listed in table 1.

Type	Methodology	Algorithm	Description
Online Learning	First Order	Perceptron [5]	The Perceptron Algorithm
		OGD [6]	Online Gradient Descent
		PA [7]	Passive Aggressive Algorithms
		ALMA [8]	Approximate Large Margin Algorithm
		RDA [3]	Regularized Dual Averaging
	Second Order	SOP [9]	Second-Order Perceptron
		CW [10]	Confidence Weighted Learning
		ECCW [11]	Exactly Convex Confidence Weighted Learning
		AROW [4]	Adaptive Regularized Online Learning
		Ada-FOBOS [12]	Adaptive Gradient Descent
		Ada-RDA [12]	Adaptive Regularized Dual Averaging
Sparse Online Learning	First Order	STG [2]	Sparse Online Learning via Truncated Gradient
		FOBOS-L1 [13]	$l1$ Regularized Forward Backward Splitting
		RDA-L1 [3]	Mixed $l1/l_2^2$ Regularized Dual Averaging
		ERDA-L1 [3]	Enhanced $l1/l_2^2$ Regularized Dual Averaging
	Second Order	Ada-FOBOS-L1 [12]	Ada-FOBOS with $l1$ regularization
		Ada-RDA-L1 [12]	Ada-RDA with $l1$ regularization

Table 1: Summary of the implemented online learning algorithms in SOL

2.2. The Software Package

The SOL package includes a library, command-line tools, and python wrappers for the learning task. SOL is implemented in standard C++ to be easily compiled and built in multiple platforms (Linux, Windows, MacOS, etc.) without dependency. It supports "libsvm" and "csv" data formats. It also defined a binary format to significantly accelerate the training process. SOL is released under the Apache 2.0 open source license.

2.2.1. Practical Usage

To illustrate the training and testing procedure, we use the *OGD* algorithm with a constant learning rate 1 to learn a model for "*rcv1*" and save the model to "*rcv1.model*".

```
$ SOL_train --params eta=1 -a ogd rcv1_train rcv1.model
[output skipped]
$ SOL_test rcv1.model rcv1_test predict.txt
test accuracy: 0.9545
```

We can also use the python wrappers to train the same model. The wrappers provide the cross validation ability which can be used to select the best parameters as the following commands show. More advanced usages of SOL can be found in the documentation.

```
$ SOL_train.py --cv eta=0.25:2:128 -a ogd rcv1_train rcv1.model
cross validation parameters: [('eta', 32.0)]
$ SOL_test.py rcv1.model rcv1_test predict.txt
test accuracy: 0.9744
```

2.2.2. Documentation and Design

The SOL package comes with detailed documentation. The README file gives an "*Installation*" section for different platforms, and a "*Quick Start*" section as a basic tutorial to use the package for training and testing. We also provide a manual for advanced users. Users who want to have a comprehensive evaluation of online algorithms and parameter settings can refer to the "*Command Line Tools*" section. If users want to call the library in their own project, they can refer to the "*Library Call*" section. For those who want to implement a new algorithm, they can read the "*Design & Extension of the Library*" section. The whole package is designed for high efficiency, scalability, portability, and extensibility.

- Efficiency: it is implemented in C++ and optimized to reduce time and memory cost.
- Scalability: Data samples are stored in a sparse structure. All operations are optimized around the sparse data structure.
- Portability: All the codes follow the C++11 standard, and there is no dependency on external libraries. We use "cmake" to organize the project so that users on different platforms can build the library easily. SOL thus can run on almost every platform.
- Extensibility: (i) the library is written in a modular way, including *PARIO*(for PARallel IO), *Loss*, and *Model*. User can extend it by inheriting the base classes of these modules and implementing the corresponding interfaces; (ii) We try to relieve the pain of coding in C++ so that users can implement algorithms in a "Matlab" style. The code snippet in Figure 1 shows an example to implement the core function of the "*ALMA*" algorithm.

Algorithm	Train Time(s)	Accuracy	Algorithm	Train Time(s)	Accuracy
Perceptron	8.4296 ± 0.0867	0.9625 ± 0.0014	OGD	8.4109 ± 0.0982	0.9727 ± 0.0006
PA	8.4506 ± 0.1031	0.9649 ± 0.0015	PA1	8.5113 ± 0.1143	0.9760 ± 0.0005
PA2	8.4445 ± 0.1068	0.9758 ± 0.0003	ALMA	9.1464 ± 0.1624	0.9745 ± 0.0009
RDA	8.4809 ± 0.0899	0.9212 ± 0.0000	ERDA	8.4623 ± 0.1123	0.9493 ± 0.0002
CW	8.4356 ± 0.1118	0.9656 ± 0.0010	ECCW	8.4641 ± 0.1116	0.9681 ± 0.0009
SOP	8.5246 ± 0.1017	0.9627 ± 0.0012	AROW	8.4390 ± 0.1292	0.9766 ± 0.0002
Ada-FOBOS	8.4897 ± 0.0872	0.9769 ± 0.0003	Ada-RDA	8.4388 ± 0.1140	0.9767 ± 0.0003
VW	11.3581 ± 0.3423	0.9754 ± 0.0009	LIBLINEAR	77.9274 ± 1.4742	0.9771 ± 0.0000

Table 2: Comparison of SOL with LIBLINEAR and VW on "*rcv1*"

```
Vector<float> w; //weight vector
void Iterate(SVector<float> x, int y) {
  //predict label with dot product
  float predict = dotmul(w, x);
  float loss = max(0, 1 - y * predict); //hinge loss
  if (loss > 0) { //non-zero loss, update the model
    w = w + eta * y * x; //eta is the learning rate
    //calculate the L2 norm of weight vector
    float w_norm = Norm2(w);
    if (w_norm > 1) w /= w_norm;
  }
}
```

Figure 1: Example code to implement the core function of "*ALMA*" algorithm.

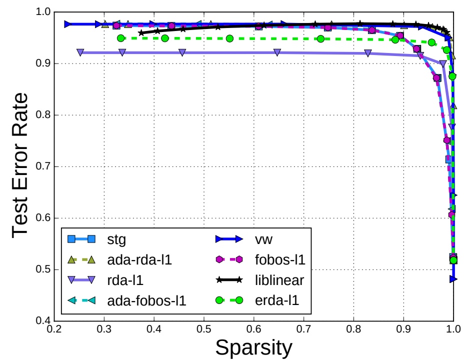

Figure 2: Comparison of Sparse Online Learning algorithms.

2.3. Comparisons

Due to space limitation, we only demonstrate that: 1) the online learning algorithms quickly reach comparable test accuracy compared to L2-SVM in LIBLINEAR [14] and VW [1]; 2) the sparse online learning methods can select meaningful features compared to L1-SVM in LIBLINEAR and L1-SGD in VW. According to Table 2, SOL provides a wide variety of algorithms that can achieve comparable test accuracies as LIBLINEAR and VW, while the training time is significantly less than LIBLINEAR. VW is also an efficient and effective online learning tool, but may not be a comprehensive platform for researchers due to its limited number of algorithms and somewhat complicate designs. Figure 2 shows how the test accuracy varies with model sparsity. L1-SVM does not work well in low sparsity due to inappropriate regularization. According to the curves, the Ada-RDA-L1 algorithm achieves the best test accuracy for almost all model sparsity values. Clearly, SOL is a highly efficient and effective online learning toolbox. More empirical results on other datasets can be found at `https://github.com/LIBOL/SOL/wiki/Example`.

2.4. Illustrative Examples

Illustrative examples of SOL can be found at: https://github.com/LIBOL/SOL/wiki/Example

[1]`https://github.com/JohnLangford/vowpal_wabbit`. VW is another OL tool with only a few algorithms

3. Conclusion

SOL is an easy-to-use open-source package of scalable online learning algorithms for large-scale online classification tasks. SOL enjoys high efficiency and efficacy in practice, particularly when dealing with high-dimensional data. In the era of big data, SOL is not only a sharp knife for machine learning practioners in learning with massive high-dimensional data, but also a comprehensive research platform for online learning researchers.

Acknowledgements

This work was done when the first author was an exchange student at Prof Hoi's research group.

References

[1] S. C. Hoi, J. Wang, P. Zhao, Libol: A library for online learning algorithms, The Journal of Machine Learning Research 15 (1) (2014) 495–499.

[2] J. Langford, L. Li, T. Zhang, Sparse online learning via truncated gradient, The Journal of Machine Learning Research 10 (2009) 777–801.

[3] L. Xiao, Dual averaging methods for regularized stochastic learning and online optimization, The Journal of Machine Learning Research 9999 (2010) 2543–2596.

[4] K. Crammer, A. Kulesza, M. Dredze, Adaptive regularization of weight vectors, Machine Learning (2009) 1–33.

[5] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain., Psychological review 65 (6) (1958) 386.

[6] M. Zinkevich, Online convex programming and generalized infinitesimal gradient ascent.

[7] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, Y. Singer, Online passive-aggressive algorithms, The Journal of Machine Learning Research 7 (2006) 551–585.

[8] C. Gentile, A new approximate maximal margin classification algorithm, J. Mach. Learn. Res. 2 (2002) 213–242.

[9] N. Cesa-Bianchi, A. Conconi, C. Gentile, A second-order perceptron algorithm, SIAM J. Comput. 34 (3) (2005) 640–668.

[10] M. Dredze, K. Crammer, F. Pereira, Confidence-weighted linear classification, in: Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 264–271.

[11] K. Crammer, M. Dredze, F. Pereira, Exact convex confidence-weighted learning, in: Advances in Neural Information Processing Systems, 2008, pp. 345–352.

[12] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research 12 (2011) 2121–2159.

[13] J. Duchi, Y. Singer, Efficient online and batch learning using forward backward splitting, The Journal of Machine Learning Research 10 (2009) 2899–2934.

[14] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, C.-J. Lin, Liblinear: A library for large linear classification, Journal of Machine Learning Research 9 (2008) 1871–1874.

Required Metadata

Current executable software version

Ancillary data table required for sub version of the executable software: (x.1, x.2 etc.) kindly replace examples in right column with the correct information about your executables, and leave the left column as it is.

Nr.	(executable) Software metadata description	Please fill in this column
S1	Current software version	v1.0.0
S2	Permanent link to executables of this version	`https://github.com/LIBOL/SOL/archive/v1.0.0.zip`
S3	Legal Software License	Apache 2.0 open source license
S4	Computing platform / Operating System	Linux, OS X, Windows.
S5	Installation requirements & dependencies	Python 2.7
S6	Link to user manual	`https://github.com/LIBOL/SOL/wiki`
S7	Support email for questions	chhoi@smu.edu.sg

Table 3: Software metadata (optional)

Current code version

Ancillary data table required for subversion of the codebase. Kindly replace examples in right column with the correct information about your current code, and leave the left column as it is.

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1.0.0
C2	Permanent link to code/repository used of this code version	`https://github.com/LIBOL/SOL/`
C3	Legal Code License	Apache 2.0 open source license
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Python/C/C++
C6	Compilation requirements, operating environments & dependencies	Python2.7/GCC/MSVC
C7	If available Link to developer documentation/manual	`https://github.com/LIBOL/SOL/wiki`
C8	Support email for questions	chhoi@smu.edu.sg

Table 4: Code metadata (mandatory)