Easy Hyperparameter Search Using Optunity

Marc Claesen MARC.CLAESEN@ESAT.KULEUVEN.BE
Jaak Simm JAAK.SIMM@ESAT.KULEUVEN.BE
Dusan Popovic DUSAN.POPOVIC@ESAT.KULEUVEN.BE
Yves Moreau YVES.MOREAU@ESAT.KULEUVEN.BE
Bart De Moor BART.DEMOOR@ESAT.KULEUVEN.BE
KU Leuven, Department of Electrical Engineering (ESAT)
STADIUS Center for Dynamical Systems, Signal Processing and Data Analytics
iMinds, Department of Medical Information Technologies
Kasteelpark Arenberg 10, box 2446
3001 Leuven, Belgium

Abstract

Optunity is a free software package dedicated to hyperparameter optimization. It contains various types of solvers, ranging from undirected methods to direct search, particle swarm and evolutionary optimization. The design focuses on ease of use, flexibility, code clarity and interoperability with existing software in all machine learning environments. Optunity is written in Python and contains interfaces to environments such as R and MATLAB. Optunity uses a BSD license and is freely available online at `http://www.optunity.net`.

Keywords: hyperparameter search, black-box optimization, algorithm tuning, Python

1. Introduction

Many machine learning tasks aim to train a model $\mathcal{M}$ which minimizes some loss function $\mathcal{L}(\mathcal{M} \mid \mathbf{X}^{(te)})$ on given test data $\mathbf{X}^{(te)}$. A model is obtained via a learning algorithm $\mathcal{A}$ which uses a training set $\mathbf{X}^{(tr)}$ and solves some optimization problem. The learning algorithm $\mathcal{A}$ may itself be parameterized by a set of hyperparameters λ, e.g. $\mathcal{M} = \mathcal{A}(\mathbf{X}^{(tr)} \mid \lambda)$. Hyperparameter search – also known as tuning – aims to find a set of hyperparameters λ^*, such that the learning algorithm yields an optimal model $\mathcal{M}^*$ that minimizes $\mathcal{L}(\mathcal{M} \mid \mathbf{X}^{(te)})$:

$$\lambda^* = \arg\min_{\lambda} \mathcal{L}\big(\mathcal{A}(\mathbf{X}^{(tr)} \mid \lambda) \mid \mathbf{X}^{(te)}\big) = \arg\min_{\lambda} \mathcal{F}(\lambda \mid \mathcal{A},\ \mathbf{X}^{(tr)}, \mathbf{X}^{(te)},\ \mathcal{L}) \tag{1}$$

In the context of tuning, $\mathcal{F}$ is the objective function and λ is a tuple of hyperparameters (optimization variables). The learning algorithm $\mathcal{A}$ and data sets $\mathbf{X}^{(tr)}$ and $\mathbf{X}^{(te)}$ are known. Depending on the learning task, $\mathbf{X}^{(tr)}$ and $\mathbf{X}^{(te)}$ may be labeled and/or equal to each other. The objective function often has a constrained domain (for example regularization terms must be positive) and is assumed to be expensive to evaluate, black-box and non-smooth.

Tuning hyperparameters is a recurrent task in many machine learning approaches. Some common hyperparameters that must be tuned are related to kernels, regularization, learning rates and network architecture. Tuning can be necessary in both supervised and unsupervised settings and may significantly impact the resulting model's performance.

© Marc Claesen, Jaak Simm, Dusan Popovic, Yves Moreau and Bart De Moor.

General machine learning packages typically provide only basic tuning methods like grid search. The most common tuning approaches are grid search and manual tuning (Hsu et al., 2003; Hinton, 2012). Grid search suffers from the curse of dimensionality when the number of hyperparameters grows large while manual tuning requires considerable expertise which leads to poor reproducibility, particularly when many hyperparameters are involved.

2. Optunity

Our software is a Swiss army knife for hyperparameter search. Optunity offers a series of configurable optimization methods and utility functions that enable efficient hyperparameter optimization. Only a handful of lines of code are necessary to perform tuning. Optunity should be used in tandem with existing machine learning packages that implement learning algorithms. The package uses a BSD license and is simple to deploy in any environment. Optunity has been tested in Python, R and MATLAB on Linux, OSX and Windows.

2.1 Functional overview

Optunity provides both simple routines for lay users and expert routines that enable fine-grained control of various aspects of the solving process. Basic tuning can be performed with minimal configuration, requiring only an objective function, an upper limit on the number of evaluations and box constraints on the hyperparameters to be optimized.

The objective function must be defined by the user. It takes a hyperparameter tuple λ and typically involves three steps: (i) training a model $\mathcal{M}$ with λ, (ii) use $\mathcal{M}$ to predict a test set (iii) compute some score or loss based on the predictions. In unsupervised tasks, the separation between (i) and (ii) need not exist, for example in clustering a data set.

Tuning involves a series of function evaluations until convergence or until a predefined maximum number of evaluations is reached. Optunity is capable of vectorizing evaluations in the working environment to speed up the process at the end user's volition.

Optunity additionally provides k-fold cross-validation to estimate the generalization performance of supervised modeling approaches. The cross-validation implementation can account for strata and clusters.[1] Finally, a variety of common quality metrics is available.

The code example below illustrates tuning an SVM with scikit-learn and Optunity.[2]

```
1 @optunity.cross_validated(x=data, y=labels, num_folds=10, num_iter=2)
2 def svm_auc(x_train, y_train, x_test, y_test, C, gamma):
3     model = sklearn.svm.SVC(C=C, gamma=gamma).fit(x_train, y_train)
4     decision_values = model.decision_function(x_test)
5     return optunity.metrics.roc_auc(y_test, decision_values)
6
7 optimal_pars, _, _ = optunity.maximize(svm_auc, num_evals=100, C=[0, 10], gamma=[0, 1])
8 optimal_model = sklearn.svm.SVC(**optimal_pars).fit(data, labels)
```

The objective function as per Equation (1) is defined on lines 1 to 5, where $\lambda = (C, \gamma)$, $\mathcal{A}$ is the SVM training algorithm and $\mathcal{L}$ is area under the ROC curve. We use $2\times$ iterated 10-fold cross-validation to estimate area under the ROC curve. Up to 100 hyperparameter tuples are tested within the box constraints $0 < C < 10$ and $0 < \gamma < 1$ on line 7.

1. Instances in a stratum should be spread across folds. Clustered instances must remain in a single fold.
2. We assume the correct imports are made and `data` and `labels` contain appropriate content.

2.2 Available solvers

Optunity provides a wide variety of solvers, ranging from basic, undirected methods like grid search and random search (Bergstra and Bengio, 2012) to evolutionary methods such as particle swarm optimization (Kennedy, 2010) and the covariance matrix adaptation evolutionary strategy (CMA-ES) (Hansen and Ostermeier, 2001). Finally, we provide the Nelder-Mead simplex (Nelder and Mead, 1965), which is useful for local search after a good region has been determined. Optunity's current default solver is particle swarm optimization, as our experiments have shown it to perform well for a large variety of tuning tasks involving various learning algorithms. Additional solvers will be incorporated in the future.

2.3 Software design and implementation

The design philosophy of Optunity prioritizes code clarity over performance. This is justified by the fact that objective function evaluations constitute the real performance bottleneck.

In contrast to typical Python packages, we avoid dependencies on big packages like NumPy/SciPy and scikit-learn to facilitate users working in non-Python environments (sometimes at the cost of performance). To prevent issues for users that are unfamiliar with Python, care is taken to ensure all code in Optunity works out of the box on any Python version above 2.7, without requiring tools like `2to3` to make explicit conversions. Optunity has a single dependency on DEAP (Fortin et al., 2012) for the CMA-ES solver.

A key aspect of Optunity's design is interoperability with external environments. This requires bidirectional communication between Optunity's Python back-end ($\mathcal{O}$) and the external environment ($\mathcal{E}$) and roughly involves three steps: (i) $\mathcal{E} \rightarrow \mathcal{O}$ solver configuration, (ii) $\mathcal{O} \leftrightarrow \mathcal{E}$ objective function evaluations and (iii) $\mathcal{O} \rightarrow \mathcal{E}$ solution and solver summary. To this end, Optunity can do straightforward communication with any environment via sockets using JSON messages as shown in Figure 1. Only some information must be communicated, big objects like data sets are never exchanged. To port Optunity to a new environment, a thin wrapper must be implemented to handle communication.

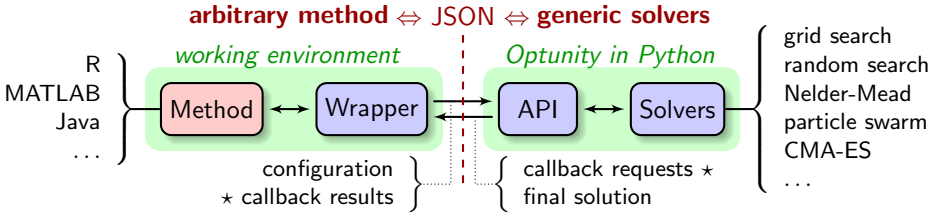

Figure 1: Integrating Optunity in non-Python environments.

2.4 Documentation

Code is documented using Sphinx and contains many doctests that can serve as both unit tests and examples of the associated functions. Our website contains API documentation, user documentation and a wide range of examples to illustrate all aspects of the software. The examples involve various packages, including scikit-learn (Pedregosa et al., 2011), OpenCV (Bradski, 2000) and Spark's MLlib (Zaharia et al., 2010).

2.5 Collaborative and future development

Collaborative development is organized via GitHub.[3] The project's master branch is kept stable and is subjected to continuous integration tests using Travis CI. We recommend prospective users to clone the master branch for the most up-to-date stable version of the software. Bug reports and feature requests can be filed via issues on GitHub.

Future development efforts will focus on wrappers for Java, Julia and C/C++. This will make Optunity readily available in all main environments related to machine learning. We additionally plan to incorporate Bayesian optimization strategies (Jones et al., 1998).

3. Related work

A number of software solutions exist for hyperparameter search. HyperOpt offers random search and sequential model-based optimization (Bergstra et al., 2013). Some packages dedicated to Bayesian approaches include Spearmint (Snoek et al., 2012), DiceKriging (Roustant et al., 2012) and BayesOpt (Martinez-Cantin, 2014). Finally, ParamILS is a command-line-only tuning framework providing iterated local search (Hutter et al., 2009).

Optunity distinguishes itself from existing packages by exposing a variety of fundamentally different solvers. This matters because the no free lunch theorem suggests that no single approach is best in all settings (Wolpert and Macready, 1997). Additionally, Optunity is easy to integrate in various environments and features a very simple API.

Acknowledgments

This research was funded via the following channels:

- Research Council KU Leuven: GOA/10/09 MaNet, CoE PFV/10/016 SymBioSys;
- Flemish Government: FWO: projects: G.0871.12N (Neural circuits); IWT: TBM-Logic Insulin(100793), TBM Rectal Cancer(100783), TBM IETA(130256), O&O ExaScience Life Pharma, ChemBioBridge, PhD grants (specifically 111065); Industrial Research fund (IOF): IOF/HB/13/027 Logic Insulin; iMinds Medical Information Technologies SBO 2014; VLK Stichting E. van der Schueren: rectal cancer
- Federal Government: FOD: Cancer Plan 2012-2015 KPC-29-023 (prostate)
- COST: Action: BM1104: Mass Spectrometry Imaging

References

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(1):281–305, 2012.

James Bergstra, Dan Yamins, and David D Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference*, pages 13–20. SciPy, 2013.

3. We maintain the following subdomains for convenience: `http://`{builds, docs, git, issues}`.optunity.net`.

G. Bradski. The OpenCV library. *Dr. Dobb's Journal of Software Tools*, 2000. URL `http://www.drdobbs.com/open-source/the-opencv-library/184404319`.

Félix-Antoine Fortin, De Rainville, Marc-André Gardner Gardner, Marc Parizeau, Christian Gagné, et al. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13(1):2171–2175, 2012.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural Networks: Tricks of the Trade*, pages 599–619. Springer, 2012.

Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

James Kennedy. Particle swarm optimization. In *Encyclopedia of Machine Learning*, pages 760–766. Springer, 2010.

Ruben Martinez-Cantin. BayesOpt: A Bayesian optimization library for nonlinear optimization, experimental design and bandits. *arXiv preprint arXiv:1405.7430*, 2014.

John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Olivier Roustant, David Ginsbourger, Yves Deville, et al. DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. 2012.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

David H Wolpert and William G Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 1–7, 2010.