Abstract: We present Optunity, a Python library which bundles various strategies to solve hyperparameter tuning problems. The library provides general purpose algorithms, ranging from undirected search methods to adaptive methods based on refinement strategies, heuristics and evolutionary computing. Optunity aspires to become a Swiss army knife to solve tuning problems of any nature. Its design focuses on code clarity, flexibility and ease of use.

Keywords: hyperparameter tuning, meta-optimization, free software, Python

1 Introduction

Machine learning tasks can be summarized as constructing a model which, in some way, captures relevant aspects of the data that was used to construct the model. The methods used in machine learning vary greatly in approach and application domain (e.g. supervised vs. unsupervised, classification vs. regression, ...), but many expose a common issue: hyperparameters which must be optimized.

Optimizing hyperparameters is referred to as tuning within the machine learning community. Commonly recurring tuning parameters are related to regularization, kernels, learning rate, ... Some important general observations: (i) the effect of tuning parameters can differ greatly, (ii) many tuning parameters have (box) constraints, (iii) the importance of a parameter can vary per problem and (iv) typically only a subset of tuning parameters significantly affect the quality of the model.

In the tuning problem, the objective function is some user defined score measure for a model built using a tuple of hyperparameters (the optimization variables). From an optimization perspective, the objective function typically is stochastic, typically not well-behaved (certainly not convex) and non-smooth. We additionally assume that evaluating the objective function is expensive, since each evaluation involves training a model and evaluating its performance. A good common example for this is k-fold cross-validation. This last assumption leads us to focus on approaches that require few evaluations (adaptive methods).

In Optunity, tuning tasks are formulated as maximization problems. Optunity is in active development. The latest version can be obtained via GitHub: https://github.com/claesenm/optunity.

2 Functional overview

Optunity provides various strategies to solve tuning problems. A complete, up-to-date overview with examples of the main features is available at: http://optunity.readthedocs.org/.

Formally, the tuning task involves optimizing a parameter vector $\theta$, such that a model $M$ trained on a data set $X$ with hyperparameters $\theta$ maximizes some score function. This can be summarized as follows:

$$\max_{\theta} f(\theta), \quad f(\theta) = \text{score}(M_{\theta}(X))$$

The choice of score function depends on the task and its design priorities (such as accuracy for classification, $R^2$ for regression). Optunity treats the objective function $f(\theta)$ as a black box. All solution strategies operate based on sequential function evaluations, without additional information like gradients or error estimates.

Optunity provides repeated k-fold cross-validation and a variety of score functions to estimate generalization performance of supervised algorithms.

2.1 Solvers

The library offers a variety of solvers, both adaptive and unadaptive. The following approaches are offered currently: grid search, random search [1], particle swarms [2] and CMA-ES [3]. Future development will incorporate DIRECT [4] and Bayesian approaches such as EGO [5] and sequential Kriging [6].

3 Implementation overview

Optunity consists of loosely coupled modules. The code is extensively documented and uses doctests where applicable, both to extend the documentation and provide unit tests. We opted to develop in Python for three key reasons: (i) terseness of the language, (ii) ease of inte-
gration with other tools and (iii) availability of other libraries (such as NumPy/SciPy [7] and DEAP [8]).

Optunity has no hard dependencies on external packages, but can benefit from some. When useful algorithms are available in other packages we can provide additional solvers.

4 Code example

The code snippet below shows how to optimize SVM hyperparameters c and g using cross-validation to estimate generalization performance on the data set \( x \mapsto y \) (data to labels). Optunity functions are shown in green.

```python
import optunity as opt

def svm_score(x_train, y_train, x_test, y_test, c, g):
    model = train(x_train, y_train, c, g)
    y_hat = predict(model, x_test)
    return some_score_fcn(y_test, y_hat)

@opt.cross_validated(x, y=y)
solver = opt.make_solver('random search', 100, c=[1, 2], g=[0.1, 10])
pars = some_pars
optimal_model = train(x, y, *pars)
```

Listing 1: tuning an SVM with Optunity

The `cross_validated` function decorator takes care of cross-validation (line 2). Calls to `svm_score` automatically yield a cross-validated estimate of generalization performance (computed using a fixed set of folds). The body of `svm_score` fits a model on the training folds and predicts the test fold. The test predictions and true test labels are used by some score function. `svm_score` is the objective function which will be maximized. Such train–predict–score chains are the only logic users need to implement to perform cross-validation.

The solver is constructed on line 7. We use random search [1], which requires a number of evaluations and a set of box constraints. The call to `maximize` couples the solver with its objective function. `maximize` returns the best parameter tuple, which can then be used to train a model on the full training set.

5 Conclusion

Optunity allows easy integration of sophisticated tuning strategies in machine learning workflows. Adaptive approaches provide disciplined ways to retrieve good parameter values while using fewer function evaluations than traditional grid search. The library provides various solving strategies and complementary functionality such as cross-validation and common score functions for supervised algorithms.

Future development goals include making Optunity available in other environments (e.g. R, MATLAB) and implementing additional state-of-the-art solvers.

Acknowledgments

Research supported by:

- Research Council KU Leuven: GOA/10/09 MaNet, KUL PFV/10/016 SymBioSys, PhD-/Postdoc grants,
- Industrial Research fund (IOF): IOF/13/027 Logic Insulin,
- Flemish Government: FWO: projects: G.0871.12N (Neural circuits); PhD/Postdoc grants; IWT: TBM-Logic Insulin (100793), TBM Rectal Cancer (100783), TBM IETA (130256); PhD/Postdoc grants; Hercules Stichting: Hercules 3: PacBio RS, Hercules 1: The C1 single-cell auto prep system, BioMark HD System and IFC controllers (Fluidigm) for single-cell analyses; iMinds Medical Information Technologies SBO 2014; VLK Stichting E. van der Schueren: rectal cancer.

References