osprey Documentation

Release 0.4$31_g5406ac6 - py2.7.egg$

Robert T. McGibbon

August 12, 2015

Contents

Osprey is a tool for practical hyperparameter optimization of machine learning algorithms. It's designed to provide a practical, **easy to use** way for application scientists to find parameters that maximize the cross-validation score of a model on their dataset. Osprey is being developed by researchers at Stanford University with primary application areas in computational protein dynamics and drug design.

CHAPTER 1

Overview

`osprey` is a command line tool. It runs using a simple *config file* which sets up the problem by describing the *estimator*, *search space*, *strategy*, *dataset*, *cross validation*, and storage for the *results*.

Related tools include and spearmint, hyperopt, and MOE. Both hyperopt and MOE can serve as backend *search strategies* for osprey.

To get started, run `osprey skeleton` to create an example config file, and then boot up one or more parallel instances of `osprey worker`.

1.1 Background

1.1.1 Theory

Osprey is designed to optimize the hyperparameters of machine learning models by maximizing a cross-validation score. As an optimization problem, the key factors here are

- very expensive objective function evaluations (minutes to hours, or more)
- no gradient information is available
- tension between exploration of parameter space and local optimization (explore / exploit dilemma)

A good, if somewhat dated overview of this problem setting can be found in Jones, Schonlau, Welch (1998) [1]. The key idea is that we can procede by fitting a **surrogate function** or **response surface**. This surrogate function needs to provide both our best guess of the function as well as our degree of belief – our uncertainty in the parts of parameter space that we haven't yet explored. Does the maxima lie over there? Then at each iteration, a new point can be selected by maximizing the **expected improvement** over our current best solution, by maximize the expected **entropy reduction in the distribution of maxima**, [2] or a similar so-called acquisition function.

`osprey` supports multiple *search strategies* for choosing the next set of hyperparameters to evaluate your model at. The most theoretically elegant of the supported methods, Gaussian process expected improvement using the MOE backend, attacks this problem directly by modeling the objective function as a draw from a Gaussian process.

[1] Jones, D. R., M. Schonlau, and W. J. Welch. "Efficient global optimization of expensive black-box functions." *J. Global Optim.* 13.4 (1998): 455-492.

[2] Hennig, P., and C. J. Schuler. "Entropy search for information-efficient global optimization." *JMLR* 98888.1 (2012): 1809-1837.

1.1.2 References

1.2 Installation

Osprey is written in Python, and can be installed with standard python machinery

1.2.1 Development Version

```
# grab the latest version from github
$ pip install git+git://github.com/pandegroup/osprey.git
```

```
# or clone the repo yourself and run `setup.py`
$ git clone https://github.com/pandegroup/osprey.git
$ cd osprey && python setup.py install
```

1.2.2 Release Version

Currently, **we recommend that you use the development version**, since things are moving fast. However, release versions from PyPI can be installed using `pip`.

```
# grab the release version from PyPI
$ pip install osprey
```

1.2.3 Dependencies

- `six`
- `pyyaml`
- `numpy`
- `scikit-learn`
- `sqlalchemy`
- `hyperopt` (recommended, required for `engine=hyperopt_tpe`)
- `MOE` (recommended, required for `engine=moe`)
- `scipy` (optional, for testing)
- `nose` (optional, for testing)

You can grab most of them with conda.

```
$ conda install six pyyaml numpy scikit-learn sqlalchemy nose
```

Hyperopt can be installed with pip.

```
$ pip install hyperopt
```

1.2.4 Getting Moe

To use the MOE search strategy, `osprey` can call MOE via two interfaces

- MOE's REST API, over HTTP
- MOE's python API

Using the MOE REST API requires that you set up a MOE server somewhere. The recommended way to do this is via the MOE docker image. See the MOE documentation for more information.

To use the MOE python API, you must install MOE on the machines you use to run osprey. The MOE documentation has some information on how to do this, but it can be tricky. An easier alternative is to use the conda binary packages that we compiled for 64-bit linux (otherwise, sorry, you're on your own).

```
conda install -c https://conda.binstar.org/rmcgibbo moe
```

See the github repo for more info on the compilation of these binaries.

1.3 Getting Started

1.4 Configuration File

`osprey` jobs are configured via a small configuration file, which is written in a hand-editable YAML markup.

The command `osprey skeleton` will create an example `config.yaml` file for you to get started with. The sections of the file are described below.

1.4.1 Estimator

The estimator section describes the model that `osprey` is tasked with optimizing. It can be specified either as a python entry point, a pickle file, or as a raw string which is passed to python's `eval()`. However specified, the estimator should be an instance or subclass of sklearn's `BaseEstimator`

Examples:

```
estimator:
  entry_point: sklearn.linear_model.LinearRegression
```

```
estimator:
  eval: Pipeline([('vectorizer', TfidfVectorizer), ('logistic', LogisticRegression())])
  eval_scope: sklearn
```

```
estimator:
  pickle: my-model.pkl   # path to pickle file on disk
```

1.4.2 Search Space

The search space describes the space of hyperparameters to search over to find the best model. It is specified as the product space of bounded intervals for different variables, which can either be of type `int`, `float`, or `enum`. Variables of type `float` can also be warped into log-space, which means that the optimization will be performed on the log of the parameter instead of the parameter itself.

Example:

```
search_space:
  logistic__C:
    min: 1e-3
    max: 1e3
    type: float
    warp: log

 logistic__penalty:
    choices:
      - l1
      - l2
   type: enum
```

1.4.3 Strategy

Three probablistic search strategies are supported. First, random search (`strategy: {name: random}`) can be used, which samples hyperparameters randomly from the search space at each model-building iteration. Random search has been shown to be significantly more effiicent than pure grid search. Example:

```
strategy:
  name: random
```

`strategy: {name: hyperopt_tpe}` is an alternative strategy which uses a Tree of Parzen estimators, described in this paper. This algorithim requires that the external package hyperopt be installed. Example:

```
strategy:
  name: hyperopt_tpe
```

Finally, `osprey` supports a Gaussian process expected improvement search strategy, using the package MOE, with `strategy: {name: moe}`. MOE can be used either as a python package installed locally, or over a HTTP REST API. To use the REST API, specify the `url` param. Example:

```
strategy:
  name: moe
  params:
    # url: http://path.to.moe.rest.api
```

1.4.4 Dataset Loader

Example:

```
dataset_loader:
  name: joblib
  params:
    filenames: ~/path/to/file.pkl
```

1.4.5 Cross Validation

Many types of cross-validation iterators are supported. The simplest option is to simply pass an `int`, which sets up k-fold cross validation. Example:

```
cv: 5
```

To access the other iterators, use the `name` and `params` keywords:

```
cv:
  name: shufflesplit
  params:
    n_iter: 5
    test_size: 0.5
```

Here's a complete list of supported iterators, along with their `name` mappings:

- `kfold`: KFold
- `shufflesplit`: ShuffleSplit
- `loo`: LeaveOneOut
- `stratifiedkfold`: StratifiedKFold
- `stratifiedshufflesplit`: StratifiedShuffleSplit

1.4.6 Trials Storage

Example:

```
trials:
  # path to a databse in which the results of each hyperparameter fit
  # are stored any SQL database is suppoted, but we recommend using
  # SQLite, which is simple and stores the results in a file on disk.
  # the string format for connecting to other database is described here:
  # http://docs.sqlalchemy.org/en/rel_0_9/core/engines.html#database-urls
  uri: sqlite:///osprey-trials.db
```

1.5 Batch Submission

Multiple `osprey worker` processes can be run simultaneously and connect to the same trials database. The following scripts might be useful as templates for submitting multiple parallel `osprey worker` s to a cluster batch scheduling system. Depending on what scheduling software your cluster runs, you can use these scripts as a jumping off point.

1.5.1 Example PBS/TORQUE Script

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -l nodes=1:ppn=16
#PBS -l walltime=12:00:00
#PBS -V

cd $PBS_O_WORKDIR
NO_OF_CORES=`cat $PBS_NODEFILE | egrep -v '^#'\|'^$' | wc -l | awk '{print $1}'`
for i in `seq $NO_OF_CORES`; do
    osprey worker config.yaml -n 100 > osprey.$PBS_JOBID.$i.log 2>&1 &
done
wait
```

1.5.2 Example SGE Script

```
#!/bin/bash
#
#$ -cwd
#$ -j y
#$ -o /dev/null
#$ -S /bin/bash
#$ -t 1-10
#$ -l h_rt=12:00:00
#$ -V

# handle if we are or are not part of an array job
if [ "$SGE_TASK_ID" = "undefined" ]; then
    SGE_TASK_ID=0
fi

osprey worker config.yaml -n 100 > osprey.$JOB_ID.$SGE_TASK_ID.log 2>&1
```

1.5.3 Example SLURM Script

```
#!/bin/bash
#SBATCH --time=12:00:00
#SBATCH --mem=4000
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=16

NO_OF_CORES=$(expr $SLURM_TASKS_PER_NODE \* $SLURM_JOB_NUM_NODES)

for i in `seq $NO_OF_CORES`; do
    srun -n 1 osprey worker config.yaml -n 100 > osprey.$SLURM_JOB_ID.$i.log 2>&1 &
done
wait
```