SurrogateBenchmarks Documentation

Release 0.0.1dev

Katharina Eggensperger

March 17, 2016

Contents:

CHAPTER 1

What you need to run Surrogate Hyperparameter Optimization Benchmarks

Surrogate Benchmark library Provides python script to build and use benchmarks

Ready-to-use benchmarks trained regression models, that can be used with the library

HPOlib To easily evaluate hyperparameter optimization experiments

CHAPTER 2

Install

2.1 Preparing a virtualenv

We recommend using a virtualenv, because (A) you can control the version of each python package, (B) installing and removing packages becomes easy as (C) you don't need sudo rights and (D) it becomes harder to mess up up the python installation on your system.

1. Get virtualenv, then load a freshly created virtualenv. (If you are not familiar with virtualenv, you might want to read more about it)

```
pip install virtualenv
virtualenv virtualSurrogates
source virtualSurrogates/bin/activate
```

2. Install `numpy`, `scipy`, `matplotlib`, as this doesn't work through setup.py.

```
easy_install -U distribute
pip install numpy==1.8.1
pip install scipy==0.14.0
pip install matplotlib
pip install scikit-learn==0.15.1
```

This may take some time. Afterwards you can verify having those libs installed with:

```
pip freeze

    argparse==1.2.1
    mock==1.0.1
    nose==1.3.4
    numpy==1.8.1
    pyparsing==2.0.3
    python-dateutil==2.3
    pytz==2014.10
    scipy==0.14.0
    six==1.8.0
    wsgiref==0.1.2
    scikit-learn==0.15.1
    matplotlib==1.4.2
```

2.2 Install the Surrogate Benchmark Library

1. **Clone the repository:**

```
git clone https://github.com/KEggensperger/SurrogateBenchmarks.git
cd SurrogateBenchmarks
```

2. Run setup.py

```
python setup.py install
```

   This will install tools, scripts and some requirements (`networkx`, `pyparsing`, and `python-daemon`). This might take a while. When your environment is ready it could/should look like this:

```
pip freeze
    Surrogates==Nan
    argparse==1.2.1
    decorator==3.4.0
    lockfile==0.10.2
    mock==1.0.1
    networkx==1.9.1
    numpy==1.8.1
    pyparsing==2.0.3
    python-daemon==1.6.1
    python-dateutil==2.3
    pytz==2014.10
    scikit-learn==0.15.1
    scipy==0.14.0
    six==1.8.0
    wsgiref==0.1.2
```

3. If the installation was successful you can run some tests. **NOTE**: Some tests will fail, if you are using different versions of *numpy*, *scipy*, and/or *scikit-learn*. This is not problematic as some of the tests only assert that you retrieve exactly the same results as me and as the numeric results only slightly differ.

```
python setup.py test
```

NOTE: If you cannot install the library, because you cannot upgrade *scikit-learn*, *numpy*, *scipy*, etc. Make sure some version of these modules is installed and uncomment the respective lines in *install_requires*.

CHAPTER 3

Run a Surrogate Benchmarks

The surrogate benchmark runs as an independent daemon process and is connected to the optimizer via a local socket. For debugging reasons it is also possible to start this process without creating a daemon. To interact with the daemon we have two scripts:

`daemonize_benchmark.py` as the name says starting a daemon, that listens to a local socket. A daemon runs till it reaches it timeout limit, which is set to 1200 sec. This means that the daemon terminates after waiting for more than 1200secs without receiving a single request. `daemonize_benchmark.py` can also be used to stop a running daemon before the timelimit is reached.

`daemon_whisperer.py` can talk to the daemon and is used to request performance predictions from the surrogate benchmark. This script implements the same interface as all the HPOlib benchmarks. It also implements a **fallback functionality*, which tries to resurrect the daemon process if it is not running.

Example 1 and 2 will show how to use these scripts

3.1 Example 1 - Starting a Surrogate Benchmark

1. Download a trained regression models, e.g. a KNN model, which uses a one-hot encoding and is trained on all data. You can get it from automl.org

```
wget www.automl.org/downloads/surrogate/onlineLDA_surrogate.tar.gz
tar -xf onlineLDA_surrogate.tar.gz
file `pwd`/onlineLDA/models/ENCODED_onlineLDA_all_KNN
MODELNAME=`pwd`/onlineLDA/models/ENCODED_onlineLDA_all_KNN
```

2. Get the corresponding file describing the searchspace: *params.pcs*

```
    file `pwd`/onlineLDA/smac_2_06_01-dev/params.pcs
    PCSFILE=`pwd`/onlineLDA/smac_2_06_01-dev/params.pcs
```

3. Start a daemon benchmark:

```
    mkdir ~/socketdir
    SOCKETDIR=~/socketdir
    daemonize_benchmark.py --surrogateData ${MODELNAME} --pcs ${PCSFILE} --socket ${SOCKETDIR}/s
```

By adding *–dry* we do not start an actual daemon, but just print a python command, which we can run to see the surrogate benchmark work

<output of previous command>

Now the surrogate benchmark is running and listens for requests on the local socket in /socketdir/something. When this works, the benchmark outputs its regression model (which is wrong for this only one model, saying it is an SVM, but it is indeed a KNN :-)) and the time left till it terminates.

4. In a different terminal, you can send a request:

```
cd ..
<new terminal>
source virtualSurrogate/bin/activate
daemon_whisperer.py --socket ~/socketdir/something --fold 0 --folds 1 --params -Kappa 0.75 -
```

Which should give you the following output:

```
Found a socket on /home/eggenspk/socketdir/something
Requesting: --fold 0 --folds 1 --params -Kappa 0.75 -Tau 512 -S 8192..........
Answer: 1416.1125104
Result for ParamILS: SAT, 0.002427, 1, 1416.112510, -1, I'm not a daemon
```

Whereas in the benchmark's terminal you will find:

```
Got a connection: <socket._socketobject object at 0x7f0c0ea8ff30> on
Received data: --fold 0 --folds 1 --params -Kappa 0.75 -Tau 512 -S 8192
CLEAN {'Tau': 512, 'S': 8192, 'Kappa': 0.75}
AFTER Unlogging:
{'Tau': 512, 'S': 8192, 'Kappa': 0.75}
{}
Requesting performance for: [0, 512, 8192, 0.75]
Encoding categorical features using a one hot encoding scheme
My answer: 1416.1125104
```

If you are missing a parameter or otherwise kill the script you might kill the surrogate process. In such a case, you need to manually delete the socket in `socketdir/something` and start over at step 3.

5. You can now play around with the surrogate and send different requests. You can follow the requests in both terminal windows. When you are finished you can either manually kill the benchmark process with `ctr+C` or send the request to stop the process:

   ..code:: bash

   daemonize_benchmark.py –socket ~/socketdir/something –stop –pcs *pwd*/onlineLDA/smac_2_06_01-dev/params.pcs

Next you can run your surrogate benchmark as a daemon process.

3.2 Example 2 - Starting a daemon

1. Again run the command from above, but without `--dry`. You won't see any output, but you can verify with `ps -ef | grep daemon_benchmark` that your daemon is running. If not you can check `${SOCKETDIR}/somethingdaemon_log.txt` for errors.

```
daemonize_benchmark.py --surrogateData ${MODELNAME} --pcs ${PCSFILE} --socket ${SOCKETDIR}/s
```

2. Now you can send the same request as before (in the same terminal):

```
daemon_whisperer.py --socket ${SOCKETDIR}/something --fold 0 --folds 1 --params -Kappa 0.75
```

Which should give you the same output as before.

3. The benchmark output can be found in `${SOCKETDIR}/somethingdaemon_log.txt`
4. To stop the daemon run

   ..code:: bash

   daemonize_benchmark.py –socket ${SOCKETDIR} –pcs space.pcs –stop

Indices and tables

- genindex
- modindex
- search