Load Prediction Software in the “Next Generation Control Centres” Research Project

Boye A. Høverstad
Dept. of Computer and Information Science, NTNU

As part of the “Next Generation Control Centres” research project, a group of two post doctors and two faculty from the Intelligent Systems group at the Department of Computer and Information Science, NTNU, studied various time series analysis models and their application to electric power load forecasting. This report is an overview and “getting started” guide to the software developed as part of that work. Although this report will hopefully ease the startup phase for anyone who wishes to use the software and/or data as a starting point for further work, direct personal communication with one of the project participants is strongly encouraged.

The main ideas and mathematical background for the software is documented in the research papers published as part of the project, most prominently in Høverstad et al. [1], and will not be reiterated here.

1 Source code

The source code we have developed is open source, and can be found on GitHub[1].

The software is mainly written in Python, and makes use of a fairly large number of third-party software packages. In particular, Numpy, Scipy and Pandas are used extensively. The process of installing the packages upon which our source code depends is documented in Section 3. At the time of reading this report, the documentation found there is almost certainly outdated, but it may nevertheless give the reader an idea of the steps necessary to get the system up and running. Several of the packages we have used are under active development, and we have experienced that backward compatibility is not always given top priority. The currently installed packages and their version numbers at the time of writing this report are given in Listing 1.

[1] `https://github.com/axeltidemann/load_forecasting`

In addition to the source code and its dependencies, data files and potentially a place to store simulation results are needed. We have used the following directory layout:

```
<base_directory>/
├── src/
│   └── sg/
│       ├── globals.py
│       ├── data/ ........................................ Data-loading modules go here
│       │   ├── <data_dir_1>/ ..................................... e.g. "bchydro"
│       │   ├── <data_dir_2>/
│       │   └── <...>/
│       ├── models/
│       └── utils/
├── data/ ........................................................ Actual data files go here
│   ├── <data_dir_1>/ ................................................ e.g. "bchydro"
│   ├── <data_dir_2>/
│   └── <...>/
└── simulations/
```

As can be seen from this directory structure, all the actual data files reside in subdirectories inside a directory named "data" at the root of the project. Each dataset is typically stored in its own directory. The directory structure is then mirrored in the `src/sg/data` directory. Here resides the code used to load the data, as well as various descriptions of the data, software used to preprocess the data, and so on.

The file `globals.py` controls the location of the main directories, and its contents may be adjusted to accomodate a different directory structure if needed.

The most interesting part of the source code is found in the `models` directory. Most of the "programs" in this directory use an evolutionary algorithm (EA) to optimize the parameters of a specific load prediction model. As such, each model has its own "program" that can be launched from the command line. For instance, in order to run an EA to optimize the parameters of the ESN model on the transmission dataset, one may type something like `python load_prediction_esn.py -bc-data` on the command line.

Several of the models are fairly resource intensive, and the software is designed to be run under MPI on a high-performance computing cluster.

Listing 1: Installed packages and their version numbers

```
> python --version
Python 2.7.3
> pip freeze
Cython==0.17.3
MDP==3.3
PySide==1.2.2
PyWavelets==0.2.2
Pyevolve==0.6rc1
Rtree==0.7.0
deap==1.0.1
ipdb==0.7
ipython==0.13.1
matplotlib==1.4.2
mock==1.0.1
mpi4py==1.3
neurolab==0.3.4
nose==1.3.4
numexpr==2.0.1
numpy==1.9.1
openpyxl==1.8.6
pandas==0.14.0
patsy==0.3.0
pyDes==2.0.1
pyGPs==1.2
pyparsing==2.0.3
python-dateutil==2.2
python-qt==0.50
pytz==2014.9
readline==6.2.4.1
rpy2==2.3.10
scikit-learn==0.15.0b2
scipy==0.14.0
six==1.8.0
statsmodels==0.5.0
tables==2.4.0
wsgiref==0.1.2
```

2 Data Files

As further described in [1], we acquired and used several data files in this project. For transmission loads, we used publicly available loads from BC Hydro in British Columbia[2].

For household loads we used AMS/AMI data from previous research projects at SINTEF Energy AS, provided to us by Nicolai Feilberg.

For the intermediate level loads, we used data from the GEFCom 2012 competition, hosted by Kaggle[3].

For the monthly/full probability distribution prediction models (which are far from complete at the time of writing), we used data from the GEFCom 2014 competition, hosted by CrowdANALYTIX[4].

During the course of the project, we also regularly downloaded weather predictions from yr.no for various locations in Norway and Europe. The motivation for this was the fact that the Norwegian Meteorological Institute offer historical weather observations through eKlima[5], but they do not have a similar offering of weather *forecasts*. This dataset was neither used nor quality assured in the project, but may be of interest by future uses, in particular in combination with AMI data from one of the Demo Norge sites (e.g. Hvaler or Steinkjer).

Due to copyright issues, we have not made any of the data publicly available, but we may provide copies upon request, provided that this is acceptable under the conditions laid down by the owners/suppliers of the original datasets.

3 Installation of the software environment

Parts of this guide are written specifically for setting up the system on kongull.hpc.ntnu.no or vilje.hpc.ntnu.no. This relates in particular to Python, numpy and scipy. On other machines, a standard Python installation should work. It must be version 2.7 or later, not 3.x.

3.1 Prerequisites

Set up directories for storing Python:

```
mkdir -p $HOME/.local $HOME/local/lib $HOME/local/include $HOME/local/bin
```

3.1.1 Loading modules

We use Intel where possible. Unload all modules except intelcomp/12.1.0 and hdf5 (module purge may do the trick).

On Kongull:

[2] `http://transmission.bchydro.com/transmission_system/balancing_authority_load_data/historical_transmission_data.htm`
[3] `https://www.kaggle.com/c/global-energy-forecasting-competition-2012-load-forecasting`
[4] `http://www.crowdanalytix.com`
[5] `eklima.met.no`

```
module load intelcomp/12.1.0
module load hdf5
```

On Vilje:

```
module load mercurial
module load mpt/2.06
module load intelcomp/13.0.1
# Python 2.7.3 depends on intelcomp/13.0.1
module load python/2.7.3
module load hdf5/1.8.10-p1
```

3.2 Mercurial compilation and installation

Loading module mercurial on Vilje automatically loads a lot of other modules (probably due to a dependency on Python 2.7.2). In order to avoid this one may install it locally:

```
# Compilation fails with multibyte output, so make sure LC_CTYPE are not UTF.
export LC_CTYPE=C
export LANG=C
python setup.py install --force --home=$HOME
```

3.3 Python compilation and installation

Get latest sqlite source code for use with Python sqlite3 module:

```
wget http://www.sqlite.org/sqlite-amalgamation-3071100.zip
unzip sqlite-amalgamation-3071100.zip
cd ~/sqlite-amalgamation-3071100
icc  -Wl,--strip-all -fPIC -O3 -shared sqlite3.c -o libsqlite3.so.7.11
mv libsqlite3.so.7.11 ~/local/lib/
cp sqlite3*.h $HOME/local/include
pushd ~/local/lib
ln -s libsqlite3.so.7.11 libsqlite3.so
popd
icc -O3 -L$HOME/local/lib -Wl,--rpath $HOME/local/lib -lsqlite3 -o sqlite3 shell.c -lpthread -ldl
```

```
mv sqlite3 $HOME/local/bin
cd ~
```

3.3.1 Getting, patching and setting up Python

Get latest 2.x version of Python:

```
wget http://python.org/ftp/python/2.7.3/Python-2.7.3.tar.bz2
tar xvfj Python-2.7.3.tar.bz2
cd ~/Python-2.7.3
```

At the time of writing, there was a bug in Python that made the ctypes build fail under the Intel compiler. See the discussion at Stackoverflow `http://stackoverflow.com/questions/3500662/intel-compiler-and-python-ctypes-libffi` and Intel `http://software.intel.com/en-us/forums/topic/303826`. There used to be a patch here: `https://github.com/hpcugent/easybuild/blob/master/easybuild/easyconfigs/p/Python/python_libffi_int128_icc.patch`. A copy of this patch is found in the same directory as this tex file.

After applying the patch:

```
./configure --prefix=$HOME/local --without-gcc --with-cxx-main=icc CC=icc CXX=icpc
make -j 12
make install
cd ~
```

3.4 Installing Virtualenv

We have put all the remaining packages in virtualenv. The setup consists of several stages:

- Download and run `virtualenv.py` with a directory as argument. We have used `~/.local`. All the programs related to virtualenv will go here, but *not* the actual virtual environments.
- Use `pip`, which is downloaded and installed by `virtualenv.py`, to install `virtualenvwrapper`.
- Use `virtualenvwrapper` to create the actual virtual environment (we use the name *smartgrid* here).
- Launch the environment by running `workon smartgrid`, and install all the remaining packages while in this environment.

If you intend to run Python with a global install of numpy, scipy or other packages, set up virtualenv with the option `--system-site-packages`.

Here is a breakdown of the the actual steps performed. If the Python you want to use in your virtual environments is the same as the one you get when typing `python` on the command line, you may skip the `VIRTUALENVWRAPPER_PYTHON` variable.

```
curl https://raw.github.com/pypa/virtualenv/master/virtualenv.py > virtualenv.py
~/local/bin/python virtualenv.py ~/.local/
~/.local/bin/pip install --install-option="--user" virtualenvwrapper
export PATH=$PATH:~/.local/bin
export VIRTUALENVWRAPPER_PYTHON=~/.local/bin/python
source .local/bin/virtualenvwrapper.sh
mkvirtualenv --system-site-packages smartgrid
```

Note that for virtualenv to work smoothly, `~/.bashrc` must be modified as described further down in this document.
Everything beyond this point assumes that the smartgrid virtual environment has been activated by typing:

```
workon smartgrid
```

3.5 Installing Numpy and Scipy

This should normally be straightforward, but it is tricky on Rocks/kongull. The text below roughly follows the guide at `http://software.intel.com/en-us/articles/numpy-scipy-with-mkl/`. Another relevant guide is: `http://www.scipy.org/Installing_SciPy/Linux`, Section *Any distribution with Intel C compiler and MKL.*
While in virtualenv smartgrid:

```
wget http://downloads.sourceforge.net/project/numpy/NumPy/1.6.1/numpy-1.6.1.tar.gz
tar xvfz numpy-1.6.1.tar.gz
cd ~/numpy-1.6.1
```

Set up for Intel compiler. Edit numpy/distutils/fcompiler/intel.py by appending to the class 'IntelEM64TFCompiler':

```
def get_flags(self):
    return ["-xhost", "-openmp", "-fp-model", "strict", "-fPIC"]
```

In numpy/distutils/intelccompiler.py I changed all references to `cc_exe` and `self.cc_exe` to be
`icc -O3 -g -fPIC -fp-model strict -fomit-frame-pointer -openmp -xhost`.
Edit site.cfg:

```
[mkl]
library_dirs = /gpfs/shareapps/apps/modulessoftware/intel/compilers/12.1.0/composer_xe_2011_sp1.6.233/mkl/lib/intel64
include_dirs = /gpfs/shareapps/apps/modulessoftware/intel/compilers/12.1.0/composer_xe_2011_sp1.6.233/mkl/include
mkl_libs = mkl_rt
lapack_libs =
```

Then, while in the numpy source directory (e.g. `~/numpy-1.6.1`):

```
rm -rf build
python setup.py config --fcompiler=intelem --compiler=intelem --cc=icc
python setup.py build_clib  --fcompiler=intelem --compiler=intelem --force
python setup.py build_ext  --fcompiler=intelem --compiler=intelem --force
python setup.py install --force
cd ~
```

Set up Scipy:

```
wget http://sourceforge.net/projects/scipy/files/scipy/0.10.1/scipy-0.10.1.tar.gz/download
tar xvfz scipy-0.10.1.tar.gz
cd ~/scipy-0.10.1
python setup.py config --fcompiler=intelem --compiler=intelem --cc=icc
python setup.py build_clib  --fcompiler=intelem --compiler=intelem --force
python setup.py build_ext  --fcompiler=intelem --compiler=intelem --force
python setup.py install --force
cd ~
```

Testing scipy:

```
pip install nose

cat <<EOF | python
> import scipy
> scipy.test('fast')
> EOF
```

At the time of writing, the test hangs, indicating a not quite successful installation. It is the test `test_axes_argument` in `~/.virtualenvs/smartgrid/lib/python2.7/site-packages/scipy/fftpack/tests/test_basic.py` that hangs on a call to `isnan` via a call to `assert_array_almost_equal` (line 480):

```
assert_array_almost_equal(swapaxes(swapaxes(y,-1,-3),
                                            -1,-2)
                          ,fftn(ikj_space))
```

Nevertheless, Numpy and Scipy seem to work fairly well for all practical purposes.

3.6 Installing packages

This lists the necessary packages. For those that are not available through pypi, make sure you get the latest version rather than blindly following the links in this document.

If any of the pips fail, set up with the usual

```
python setup.py build
python setup.py install
```

These are the necessary packages:

```
# On Kongull:
pip install readline
pip install ipython
# On Vilje:
pip install --upgrade --force-reinstall ipython

pip install MDP
(or pip install http://sourceforge.net/projects/mdp-toolkit/files/mdp-toolkit/3.2/MDP-3.2.tar.gz/download)
# As of this writing, the released Oger (1.1.3) is buggy, it has some incorrect
tests in nodes/flows.py that will make the ESN code fail. Clone from hg instead
and set PYTHONPATH accordingly.
# wget http://organic.elis.ugent.be/...
# tar xvfz Oger-xxx.tar.gz
# cd Oger-xxx
# python setup.py install
# cd ..
hg clone https://bitbucket.org/benjamin_schrauwen/organic-reservoir-computing-engine/ ~/smartgrid/Oger
pip install numexpr
pip install cython
pip install matplotlib
pip install PyWavelets
pip install pandas
```

3.6.1 Installing the R toolchain

In order to run the AR and ARIMA prediction models (but not the AR_indexed model), R must be installed and available on the system, together with the rpy2 module.

Installing R on Vilje:

```
wget <latest R>
tar xvfz R-xxx
pushd R-xxx
./configure --prefix=$HOME/local CXX=icc CC=icc --enable-R-shlib --with-blas --with-lapack --enable-static-lib
nice make -j 12
make install
make install-libR #installs to $HOME/local/lib64
popd
pip install rpy2
```

Installing R on Kongull:

R in turn requires GNU readline. This is a different library than the Python readline module that is used by iPython.

```
wget ftp://ftp.cwru.edu/pub/bash/readline-6.2.tar.gz
tar xvfz readline-6.2.tar.gz
pushd readline-6.2
./configure --prefix=$HOME/local CC=icc
nice make -j 12
make install
popd
```

```
wget http://cran.uib.no/src/base/R-2/R-2.15.1.tar.gz
tar xvfz R-2.15.1.tar.gz
pushd R-2.15.1
./configure --prefix=$HOME/local --enable-R-shlib --enable-R-static-lib \
  CXX=icc F77=ifort CFLAGS=-I$HOME/local/include LDFLAGS="-L$HOME/local/lib \
  -Wl,--rpath=$HOME/local/lib"
nice make -j 12
make install
# Libraries must be copied manually for some reason. make install-libR doesn't work.
```

```
cp lib/libR* ~/local/lib
popd
```

After installing R, the package “forecast” must be installed. Launch R and run the following commands (a window will pop up asking the user to select CRAN mirror):

```
install.packages("forecast")
q()
```

Installing rpy2. This requires that `LD_LIBRARY_PATH` is set correctly (see `~/.bashrc` section below):

```
wget http://sourceforge.net/projects/rpy/files/rpy2/2.2.x/rpy2-2.2.3.tar.gz/download
tar xvfz rpy2-2.2.3.tar.gz
pushd rpy2-2.2.3
```

Modify setup.py as described in `http://stackoverflow.com/questions/1676384/how-to-pass-flag-to-gcc-in-python-setup-py-script`: change `include_dirs` on line 341 at the time of writing to hold the readline include directory. Given the prefix used when installing readline above, this means setting `include_dirs = <HOME>/local/include`:

```
340,342c340,342
<
<     include_dirs = []
<
---
>     prefix = os.path.join(os.environ['HOME'], 'local')
>     include_dirs = [os.path.join(prefix, 'include')]
>
```

Install as usual after the modification:

```
python setup.py build
python setup.py install
popd
```

3.6.2 Installing Pyevolve

There is a bug in the parallel version of Pyevolve. Replace GPopulation.py with “our” version (found in the same directory as this .tex file). The modified version maintains a single process pool rather than creating new processes on each generation. Check for changes against the latest Pyevolve version before replacing the file.

```
wget http://sourceforge.net/projects/pyevolve/files/Pyevolve/0.6rc1/Pyevolve-0.6rc1.tar.gz/download
tar xvfz Pyevolve-0.6rc1.tar.gz
mv GPopulation.py Pyevolve-0.6rc1/pyevolve/GPopulation.py
cd Pyevolve-0.6rc1
python setup.py install
cd ..
```

3.6.3 Installing PyTables and HDF5 support

PyTables probably obsolete with Pandas. HDF5 not necessary on Vilje.

We use PyTables ("tables") for HDF5 access. Kongull has HDF5 1.6.x, which is, at the time of writing, not supported by the latest version of tables. Current tables uses the HDF5 1.8 API.

While HDF5 1.6.x is the only option:

```
module disp hdf5
export HDF5_DIR=$HDF5ROOT
wget http://sourceforge.net/projects/pytables/files/pytables/2.3.1/tables-2.3.1.tar.gz
tar xvf tables-2.3.1.tar.gz
cd tables-2.3.1
python setup.py install
cd ..
```

For systems where HDF5 1.8 is available:

```
module disp hdf5
export HDF5_DIR=$HDF5ROOT
pip install tables
```

3.6.4 Installing libspatialindex and RTree

The CBR prediction model relies on libspatialindex and RTree.

```
wget <latest version of libspatialindex>
tar xvfj spatialindex-src-1.*.tar.bz2
cd spatialindex-src-1.*
export CC=icc
export CXX=icpc
```

```
./configure --prefix=$HOME/local
nice make -j
make install
# Make sure LD_LIBRARY_PATH is ok.
pip install Rtree
```

3.7 Setting up shell resource script

`~/.bashrc`, `~/.profile`, `~/.bash_profile` or equivalent must be modified to support the various packages and programs. Here is a list of suggestions

```
# Set PATH to find local Python, R, etc
export PATH="$HOME/local/bin:$PATH"

export LC_LANG=C
export LC_CTYPE=C

# LD_LIBRARY_PATH needed by rpy2 in order to find GNU readline.
export LD_LIBRARY_PATH="$HOME/local/lib:$LD_LIBRARY_PATH"

PYTHONPATH=$HOME/smartgrid/Oger/src/:$PYTHONPATH
PYTHONPATH=$HOME/smartgrid/src:$PYTHONPATH
PYTHONPATH=`echo $PYTHONPATH | sed -e's/:$//'`
export PYTHONPATH

# # Kongull modules (must load BEFORE virtualenv)
# module load intelcomp/12.1.0
# module load hdf5
# Vilje modules
module load intelcomp/13.0.1
# Python 2.7.3 depends on intelcomp/13.0.1
module load mpt/2.06
module load python/2.7.3
module load hdf5/1.8.10-p1
```

```
# Python virtualenv stuff
export PATH=$PATH:$HOME/.local/bin
export VIRTUALENVWRAPPER_PYTHON=$HOME/.local/bin/python
export WORKON_HOME=$HOME/.virtualenvs
source $HOME/.local/bin/virtualenvwrapper.sh
```

3.8 PBS job scripts

When submittion jobs to PBS, the environment must be set up in a similar fashion. `LD_LIBRARY_PATH` must be set in order for the Python sqlite-module to find libsqlite.

Virtualenv can run into trouble when many processes launch the same virtual environment at the same time. Therefore we start our job scripts with a short pause, in order to reduce the risk of collisions.

```
# Set up Python virtual environment. Virtualenv doesn't play well in parallel,
# so reduce the risk of errors due to synchronized calls to 'workon' by
# sleeping for a brief random period before calling workon.
seconds=`awk "END {srand($RANDOM); print int(rand()*5)}" /dev/null`
echo "Sleeping for $seconds second(s) before activating the virtual environment"
sleep $seconds
echo 'Go!'
source $HOME/.bashrc
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH":$HOME/local/lib
workon smartgrid
```

3.9 Emacs

Although not directly related, the following options can be used to compile a decent Emacs:

```
./configure --prefix=$HOME/local --without-pop --without-sound --without-xpm \
  --without-jpeg --without-tiff --without-gif --without-png --without-rsvg
```

References

[1] Boye A. Høverstad, Axel Tidemann, Helge Langseth, and Pinar Öztürk. Short term load forecasting with seasonal decomposition using evolution for parameter tuning. *IEEE Transactions on Smart Grid*, in press 2015.