Hands-on exercises with Sumatra

Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date: 2013-03-28

This worksheet will introduce you to the most important capabilities of Sumatra. To learn more, check out the documentation at http://neuralensemble.org/sumatra/. If you have questions, you will get answers in the sumatra-users Google Group at https://groups.google.com/forum/#!forum/sumatra-users.

As our example code, we will use a Python program for analyzing scanning electron microscope (SEM) images of glass samples. This example was taken from an online SciPy tutorial at http://scipy-lectures.github.com/intro/summary-exercises/image-processing.html

Get the example code

To run the example code, you will need to have the following Python packages installed: numpy, scipy, matplotlib, mercurial.

Now get the code using:

```
$ hg clone https://bitbucket.org/apdavison/ircr2013 sumatra_exercise
$ cd sumatra_exercise
```

Let's first run the computation without Sumatra:

```
$ python glass_sem_analysis.py default_parameters MV_HFV_012.jpg
1699.875 65.0
```

This program reads the parameter file `default_parameters` and the image file `MV_HFV_012.jpg`. It crops the image to remove the black bar at the bottom, filters the image, defines thresholds that distinguish between the different phases in the image (sand grains, glass, bubbles), creates a new image with each phase coloured differently, "cleans" the image and creates a figure showing the before and after cleaning images. It then calculates some simple statistics on the bubble size.

Take a look at the `Data` subdirectory. It should contain another subdirectory labelled with today's date, which in turn contains three image files.

Set up a Sumatra project

If you haven't already installed Sumatra, check out the installation instructions at http://pythonhosted.org/Sumatra/installation.html.

In the `sumatra_exercise` directory, run the following:

```
$ smt init -d Data -i . ProjectGlass
```

This creates a project with the name "ProjectGlass", and tells Sumatra that the output data files will be created within the "Data" directory and that the input data files will be created in the current directory.

Now we can run the `glass_sem_analysis.py` script again, this time with Sumatra:

```
$ smt run -e python -m glass_sem_analysis.py default_parameters MV_HFV_012.jpg
```

The "-e" stands for "executable" (you could also have used the long form: `--executable=python` and the "-m" stands for "main file". To avoid having to specify the executable and main file every time, we can set these as defaults:

```
$ smt configure -e python -m glass_sem_analysis.py
```

after which we need only specify the parameter file and input image:

```
$ smt run default_parameters MV_HFV_012.jpg
```

Getting help

To get information about any of the "smt" commands, use `smt help`, e.g.:

```
$ smt help configure
```

To get a list of all available commands:

```
$ smt
```

Try looking at the help information on some of the other commands. To see the current configuration of the project:

```
$ smt info
```

Reviewing previous computations

Let's take a look at the information Sumatra has captured about the analyses we've just run:

```
$ smt list
```

This gives a list of timestamps, each of which is an automatically-generated label for one of the computations we've run. You can also define the label yourself rather than have it be automatically generated. Run:

```
$ smt run -l example_label default_parameters MV_HFV_012.jpg
```

and then take another look at the output of `smt list`. To get more detail, use the "long" option:

```
$ smt list -l
```

This gives a lot more information for each time we ran `smt run`:

- when it was run
- how long the computation took
- which program (name, location and version)
- which script (name, version control repository, version)
- parameters
- input data files
- results

Sumatra stores still more detail, but to look at it you will need to use a web browser. Run:

```
$ smtweb
```

This launches a small webserver on your machine, and should open a new tab in your default web browser. If it does not open a tab, open one yourself and navigate to http://127.0.0.1:8000.

The first page shows just a list of projects. Of course, we have only one so far. Click on the project name. This gives a list of all the computations we've performed so far. Click on one of the labels. This shows all the details that have been captured by Sumatra: all the information shown by `smt list -l`, and in addition:

- the name, location and, where possible, the version of all of the dependencies (NumPy, matplotlib, etc.) of our analysis script.
- information about the computing platform (processor architecture, operating system)
- all of the output printed by our program to the console.

In addition, you should be able to click on the filenames of the output data files (the images generated by the `glass_sem_analysis.py` script) to see the contents of the files.

Note also that next to each filename is a long hexadecimal number. This is the SHA1 hash of the file contents, a unique summary of the file contents. If the file is changed for any reason (e.g. accidentally overwritten, corrupted), the SHA1 hash will also be changed, and this change will be detected by Sumatra.

You may have other tools on your computer for calculating SHA1 hashes. For one of the image files listed, try typing:

```
$ shasum Data/20130329/MV_HFV_012_101808_sand.png
```

(replace the timestamp in the filename with the correct one on your computer). If you don't get a "command not found" message, then this will print the SHA1 hash of the file contents. Check that it matches what is shown in the browser.

Adding further context

You will notice that the record contains empty text boxes labelled "Reason", "Outcome". This lets you record additional information to help you understand how this computation fits into your overall project: why did you run this particular analysis with these particular parameters? What, in qualitative terms, was the outcome - was it what you expected, was it a surprise, or does it seem as though there is a bug in your program? This information may seem obvious to you now, but in six months time when writing a paper or responding to reviews, you might well have forgotten the details.

Try entering some information into those boxes now; don't forget to click "Save changes" when you're done.

You can also edit this information from the command line. When you launch a computation, use the `-r` option to enter a reason:

```
$ smt run -r "test adding reason from command line" default_parameters \
                                                              MV_HFV_012.jpg
```

And afterwards, use the `smt comment` command to comment on the outcome:

```
$ smt comment "works fine"
```

The browser page also has a box for adding tags to a record, which can be extremely useful in finding the record later, or in grouping related records.

Go ahead and add some tags (separated by commas) to a few records. Now if you return to the main project page (click on "ProjectGlass") you will see the tags listed in the table and can use the Tag icon in the menu bar to filter the list of records based on tags.

You can also add tags on the command line:

```
$ smt tag mytag 20130329-101802 20130329-101707
```

(replace the labels by actual labels from your project) and filter the output of `smt list` based on tag:

```
$ smt list mytag
```

Working with parameters

Take a look at the file `default_parameters`. Let's try changing some parameters:

```
$ cp default_parameters no_filter
```

Now edit the file `no_filter` in a text editor, and change *filter_size* to 1.

```
$ smt run -r 'No filtering' no_filter MV_HFV_012.jpg
```

In the web browser, refresh the "ProjectGlass" page so that the most recent run is shown in the list, then click somewhere in the row for that record (not on the label) and drag the mouse pointer to also highlight the previous record. Now click on "compare records", which will give you a side-by-side view of the two records, allowing you to easily see how changing the filter size has changed the results.

You can also change parameters from the command line. Try:

```
$ smt run -r 'Trying a different colourmap' default_parameters \
                                        MV_HFV_012.jpg phases_colourmap=hot
```

and again do a side-by-side comparison.

```
$ smt comment 'The default colourmap is nicer'
```

Changing your code

The output printed by the `glass_sem_analysis.py` is not very useful. Let's add some labels. Open the file in a text editor and replace:

```
print mean_bubble_size, median_bubble_size
```

with:

```
print "Mean:", mean_bubble_size
print "Median:", median_bubble_size
```

Now let's run it again:

```
$ smt run -r 'Added labels to output' default_parameters MV_HFV_012.jpg
```

Whoops! Instead off running the analysis, Sumatra just prints a message "Code has changed, please commit your changes". This is because we didn't commit our changes to version control. Sumatra ensures that the precise code used for a given computation is always known. Since our code is different to the current version in the Mercurial repository, it refuses to run. Let's try again:

```
$ hg commit -m 'Added labels to output'
$ smt run -r 'Added labels to output' default_parameters MV_HFV_012.jpg
```

Now it works, and we can see in the browser interface that the version is different.

If you don't want to commit every small change, you can also ask Sumatra to store the differences for you. Make another change to the Python script. Then run:

```
$ smt configure --on-changed=store-diff
$ $ smt run -r 'made a change' default_parameters MV_HFV_012.jpg
```

This time, although the code has changed, Sumatra stores the diff for you. Refresh the web browser and note that the version number for the most recent record has an asterisk next to it. If you click through to the record detail, there is a link "diff" which will show you the un-committed changes.

Next steps

This exercise should have demonstrated the basics of using Sumatra to track a computation-based project. Note that although we have used Python in this exercise, `smt` should work with almost any command-line-based program (although tracking dependencies requires a language-specific plugin).

If you wish to explore further, the documentation is at http://neuralensemble.org/sumatra/. In particular, you may wish to explore using Sumatra in conjunction with LaTeX or Sphinx, to include figures generated by Sumatra-tracked computations in your papers or blog posts, with automatic checking of SHA1 hashes and hyperlinks to the Sumatra record for the computation that generated the figure.

Bugs and feature requests

If you find a bug in Sumatra or wish to suggest an improvement or a new feature, please open a ticket in the issue tracker at http://neuralensemble.org/trac/sumatra/

If you fix the bug or implement the improvement yourself, you can either attach a patch to the ticket, or clone the Sumatra mirror on BitBucket (https://bitbucket.org/apdavison/sumatra) and open a pull request.