

Inspect Advanced Tutorial

Run it like a pro

Sam Payne

Winter 2007

I'm opposed to tutorials that are too long or cover too much. This is why the first tutorial covered only installation and a simple search. I intentionally left out some features that we regularly use in our lab. Not because I'm mean, but rather to introduce them in an expedient manner. In this tutorial I will introduce more features involved in:

- Documentation
- Database
- Modification search
- Supercomputing
- Post-processing
- Bug reporting

as they relate to our default work-flow. 1. create a database with decoy sequences; 2. create the input file with cool parameters; 3. run Inspect; 4. Pvalue filter on the results; 5. Summary and Label to view the results. Further information and documentation are available online at <http://peptide.ucsd.edu>

Part 1: Documentation

It is only possible to explain so much in the tutorials, after which you can continue to learn by reading the documentation. We've worked hard to make documentation useful; give it a try. Files in the "Inspect/docs" folder explain various features of Inspect. The python scripts all come with embedded documentation. Typing the script name without any arguments prints the usage information to the screen. Like so

```
> python Summary.py
```

For your own edification, you should view the usage information for every script that you use.

Part 2: Database

The most glaring omission from the first tutorial was an incomplete protein database. Foreign proteins contaminate every mass spectrometry experiment. Human handling guarantees keratin contamination. Additionally, the protease that you use for digestion will be in the mix. The file "CommonContaminants.fasta," includes sequences for keratin and trypsin. Feel free to add other sequences to this file as needed. To search this database (in addition to the target database), add another line in the input file.

```
db,CommonContaminants.trie
```

Another change to the database is the inclusion of fake, or decoy, protein sequences. The reality of any MS/MS experiment is that a high percentage of spectra are junk. If your database includes 50% fake sequences, then there is a 50% chance that a junk annotation will hit a fake sequence, which helps to filter false-positive annotations. I use the suffix "RS" to denote a database with both Real and Shuffled sequences. The arguments used below will make a shuffled decoy database. If you prefer a reverse decoy, read the documentation for the script (see part 1).

```
> python ShuffleDB.py -r Database\myDB.trie -w Database\myDB.RS.trie
```

Alter the input file to reflect the new databases:

```
spectra,Fraction01.mzxml
instrument,ESI-ION-TRAP
protease,Trypsin
db,myDB.RS.trie
db,CommonContaminants.RS.trie
# Protecting group on cysteine:
mod,57,C,fix
```

Part 3: Modification Search

The new rage in proteomics is to find post-translational modifications (PTMs) on your proteins: phosphorylation, methylation, etc. Tandem MS is the perfect tool, because these modifications shift m/z peaks in the spectrum. Adding lines like those below will enumerate PTMs for searching.

```
mod,14,KR
mod,-17,CQ,nterminal
mod,80,STY,opt,phosphorylation
mods,1
```

The first line denotes an addition of 14 daltons on arginine or lysine, a methylation. The second line is an example of common *in vitro* chemical damage. Next is a phosphorylation of serine, threonine, or tyrosine. It is imperative that searches for phosphorylation include this exact line. Inspect has a specific scoring model for phosphopeptides that is much more sensitive than just a +80 modification. The last line specifies how many PTMs to allow on a peptide. It may be tempting to search for multiple PTMs per peptide. In our experience this leads to poor quality results and requires much more work in post-processing. So we typically allow only one or two. To learn about unrestrictive PTM searches, see the separate tutorial.

Part 4: Supercomputing

For large scale proteomics, supercomputing should be on your mind. Even though Inspect is faster than other MS/MS programs, it is still nice to use a grid when your dataset exceeds 200,000 spectra. Unfortunately we cannot provide scripts that will work on every grid; supercomputers differ radically in their setup. We suggest that you meet with an administrator and work out a pipeline that allows you to submit jobs to multiple processors.

Part 5: Post processing

As I did in the first tutorial, I emphasize the necessity of post-processing. Read this section carefully and follow it. First, let's learn a bit more about the flexibility of the PValue script. Remembering part 1, print the usage information for the PValue.py script. You should be excited by all the new options. The first option we will discuss is “-S” which sets the percentage of the database that is fake protein sequences. This option allows the program to create an empirical pvalue based on the distribution of scores of false hits. This method of filtering is much preferred. We never run anything in the lab without a fake database and the empirical pvalue filter. Other notable options include writing out the score distribution (-s, -i) or loading in a distribution (-l), which is

sometimes useful. Finally, the “-1” (one, not lowercase L) option ensures that only the top hit for a spectrum is included in the output.

As discussed in the first tutorial, the summary script outputs the proteins found by the search. What constitutes a “found protein” you might ask. Well, the usage information describes two criteria for inclusion. Defaults exist, but a more stringent cutoff on the number of peptides per protein (-e) or the number of spectra per protein (-m) can increase the specificity of your results.

A new script to introduce is Label.py, which creates a picture of the spectrum showing the annotated peaks. You can check out alternate annotations, e.g. changing the PTM placement or mass. To learn about it, print the usage info.

Part 6: Bug reporting

When the program breaks, what do you do? We use a bug tracking system called JIRA. Logging the bug into this system is the best way to get the developers attention, and it ensure that the bug won’t be lost in our massive email inboxes. Email spayne@ucsd.edu to get an account for the system.