Scientific Computing for Biologists

Hands-On Exercises

Lecture 12: Building a Bioinformatics Pipeline, Part II

Paul M. Magwene

22 November 2011

Bioinformatics tools

In last weeks class we worked through some examples that illustrated how standard Unix command line tools could be chained together into simple pipelines to analyze genome sequence information. Today we'll examine how to build pipelines using a mix of standard bioinformatics software packages and a Python script.

Installing the tools

For the purposes of these exercises we will install all of our software tools into a directory called `tmp` in your home directory (you might already have such a directory).

```
$ mkdir ~/tmp
```

Recall that the tilde (~) is Unix shorthand for your home directory (`/Users/pmagwene` in my case).

Clustalw

Clustalw is a popular progressive multiple sequence alignment program. There are pre-built binaries of Clustalw available but for the purposes of this exercises we're going to build the program from source. This assumes you have the appropriate command line tools available on your platform, either the Xcode tools on OS X, or Cygwin plus gcc, make, autoconf, etc.

Use the `curl` command to download the source code for Clustalw as I demonstrate below:

```
$ cd ~/tmp
$ curl -O ftp://ftp.ebi.ac.uk/pub/software/clustalw2/2.1/clustalw-2.1.tar.gz
```

Once you have the clustalw source code decompress the tarball and make the programs as follows:

```
$ tar xvzf clustalw-2.1.tar.gz
$ cd clustalw-2.1/
$ ./configure  # configures the source code for your operating system
$ make  # compiles the code files
$ sudo make install # puts the binary executables in standard directories
                    # you don't need sudo on cygwin
```

The command `sudo make install` will install the clustalw binaries you just built into the `/usr/local/bin` directory. `sudo` let's you run this command with 'superuser' (i.e. admin) privileges. Assuming everything compiled without errors you can check that the program installed correctly as follows:

```
$ which clustalw2
/usr/local/bin/clustalw2

$ clustalw2

 **************************************************************
 ******** CLUSTAL 2.0.12 Multiple Sequence Alignments  ********
 **************************************************************

     ... output truncated ...
```

MAFFT

MAFFT is another multiple sequence alignment program. It's relatively fast and a number of studies have shown that it is amongst the best performing multiple sequence aligners. MAFFT is usually the sequence aligner I reach for first. Clustalw is the 'classic' alignment tool, so it's useful to have on your system, but MAFFT usually gives better alignments (though Clustalw2 is supposed to address some of the short-comings of the older versions of Clustalw). See the MAFFT website for additional references and information.

There are pre-compiled MAFFT binaries available on the MAFFT website. However, again let's build this program from source. First, we grab the latest version of the source code (as of 10 Nov 2011):

```
$ cd ~/tmp; curl -O http://mafft.cbrc.jp/alignment/software/mafft-6.864-with-extensions-
    src.tgz
```

Notice how I put two commands on the same line by separating them with a semi-colon. We then unpack the tarball and build the source as follows:

```
$ cd mafft-6.864-with-extensions/core
$ make clean
$ make
$ sudo make install
```

Building MAFFT was almost exactly the same as building Clustalw, except we didn't need to use the `configure` command in this case.

Once you've built and installed MAFFT check the installation location and confirm that the binary is working:

```
$ which mafft
/usr/local/bin/mafft
$ mafft  # type ctrl-c to exit from the interactive prompt

---------------------------------------------------------------------

   MAFFT v6.864b (2011/11/10)

        Copyright (c) 2011 Kazutaka Katoh
        NAR 30:3059-3066, NAR 33:511-518
        http://mafft.cbrc.jp/alignment/software/
---------------------------------------------------------------------
```

HMMER

HMMER is aa implementation of a profile Hidden Markov Model (HMM) for protein sequence analysis. You can read up on HMMER at the HMMER website. We will use it here for finding protein domains in sequences in conjuction with the PFAM database. By now, the steps needed to download and build programs from source are familiar to you:

```
$ cd ~/tmp
$ curl -O http://selab.janelia.org/software/hmmer3/3.0/hmmer-3.0.tar.gz
$ tar zxf hmmer-3.0.tar.gz
$ cd hmmer-3.0
$ ./configure
$ make
$ make check # optional, not every source package includes such a call
$ sudo make install
```

After compiling from source as above the binaries will be placed in `/usr/local/bin`.

Get the PFAM HMM library

We will be using the Pfam database (Release 25) in conjunction with HMMER to search for known protein domains in our sequences of interest. Since the the Pfam HMM libraries are large I'll try and provide a couple of thumb drives with the necessary library. If you're using this document outside of class you can download the necessay library as follows:

```
$ curl -O ftp://ftp.sanger.ac.uk/pub/databases/Pfam/releases/Pfam25.0/Pfam-A.hmm.gz
```

This is a large file (185MB) and decompresses to an even larger file (approx. 986MB). Make sure you have adequate disk space. Uzip it as follows:

```
$ gunzip Pfam-A.hmm.gz
```

Install the BLAST+ suite

NCBI provides a set of command line blast tools called the BLAST+ suite. This used to include a command line program called `blastcl3` that you could use to run a BLAST search against the NCBI BLAST servers. In the latest version of the BLAST+ toolset each of the core blast programs has a `-remote` option that does the same thing. The `-remote` option is useful for a moderate number of files, but if you were building a computationally intensive pipeline you would want to install a local copy of the BLAST databases and the accompanying tools.

Becuase the BLAST+ tools are complicated to build from source, we'll use the precompiled binaries that NCBI provides. Download the appropriate binary distribution for your from the NCBI ftp server (see links on class wiki). Once you've got the file you simply unzip the tarball in your `~/tmp` directory. The binaries can be installed elsewhere but we'll just leave everything in `~/tmp`.

```
$ cd tmp
$ curl -O ftp://ftp.ncbi.nlm.nih.gov/blast/executables/LATEST/ncbi-blast-2.2.25+-universal
    -macosx.tar.gz
$ tar xvzf ncbi-blast-2.2.25+-universal-macosx.tar.gz
```

To confirm that the binary works on your system do the following:

```
$ ./ncbi-blast-2.2.25+/bin/blastp -help
```

If you're on a Mac and get an error like the one shown below, try downloading and installing a slightly older version of the BLAST+ tools from `ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/2.2.23/`.

```
dyld: lazy symbol binding failed: Symbol not found:
    __ZSt16__ostream_insertIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_PKS3_i
  Referenced from: /Users/pmagwene/tmp/./ncbi-blast-2.2.24+/bin/blastp
  Expected in: flat namespace

dyld: Symbol not found:
    __ZSt16__ostream_insertIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_PKS3_i
  Referenced from: /Users/pmagwene/tmp/./ncbi-blast-2.2.24+/bin/blastp
  Expected in: flat namespace
```

Once you've confirmed that `blastp` works on your system check out the PDF documentation in the directory `~/tmp/ncbi-blast-2.2.25+/doc/` (or wherever you unzipped the file).

Install Biopython

If you're on Windows you can download a prebuilt binary of Biopython from the Biopython website. For OS X you can install Biopython from source relatively easily, assuming you've previously installed setuptools which includes the `easy_install` program:

```
sudo easy_install -f http://biopython.org/DIST/ biopython
```

Testing the bionformatics tools

Download the `fungal-ras.fas` file from the class website. This file includes protein sequences of Ras-family proteins from a number of different fungi. Ras proteins are small GTPases that are involved in cellular signal transduction. Ras signaling is often involved in cellular growth and differentiation and mutations that affect Ras signaling often lead to cancer. We'll use this data to do some quick tests to confirm that our software tools are working correctly. Of course, when putting together an analysis pipeline for your own purposes you'll want to spend a fair amount of time reading the documentation (and related papers) for each tool and make sure you understand the various options and settings.

Multiple alignments with Clustalw and MAFFT

```
$ clustalw2 -infile=fungal-ras.fas
$ mafft --auto fungal-ras.fas > fungal-ras-mafft.fas
```

The commands above should produce the following files: `fungal-ras.aln` and `fungal-ras.dnd` (Clustalw) and `fungal-ras-mafft.fas` (MAFFT). To visualize the alignments there are a variety of different multiple alignment viewers. One such program is Jalview, a free cross platform, multiple alignment viewer/editor written in Java. Take a look at both the clustalw and mafft alignments using Jalview.

Testing HMMER

To test out HMMER and Pfam download the `Rme1.fas` file from the class website. Rme1 is a transcription factor that regulates sporulation and meiosis in budding yeast, *Saccharomyces cerevisiae*. We'll use HMMER to analyze the domain structure of Rme1.

The first thing you'll need to do is run `Pfam-A.hmm` through the `hmmpress` program which prepares the HMM database for fast scanning by creating binary files. This might take a few minutes depending on the speed of your machine.

```
$ hmmpress Pfam-A.hmm
```

This will create a number of additional files in the same directory as `Pfam-A.hmm`. We can now use `hmmscan` to search for known protein domains included in the Pfam database.

```
$ hmmscan Pfam-A.hmm Rme1.fas | less
```

The default output from `hmmscan` is designed to be human readable. For outputs that are easier to parse computationally use the `--tblout` or `--domtblout` options to save output in a tabular format.

```
$ hmmscan --domtblout Rme1-output.txt -o /dev/null Pfam-A.hmm Rme1.fas
```

This call produces a file `Rme1-output.txt` that contains a space delimited text file summarizing the per-domain output. The `-o` option redirects the main human-readable output (in this case to the 'bit-bucket', /dev/null). You can then manipulate the tabular output in `Rme1-output.txt` using standard Unix tools like `awk`.

See pp. 24-26 of the HMMER user guide for more info on the `hmmscan` program and settings. E-values and bit-scores are the criteria you want to look at when trying to judge which domains you sequence of interest has good matches to. HMMER bit scores reflect the extent to which a sequence is a good match to a profile model (higher bit scores are better matches). See p. 43 of the HMMER 2.3 user guide (use Google to find a copy of the older version of the HMMER manual) for a discussion of E-values and bit scores. If you examine the output file `Rme1-output.txt` you'll see that the model with the lowest E-values and highest bit-scores is a "zinc finger" domain. There are three such domains in the Rme1 protein (see the column labeled "N" in the output), though two of them are weaker matches (larger E-values). Rme1 is a zinc finger transcriptional factor. The two weaker zinc finger domains are weak matches to the HMM model for zinc fingers but are nonetheless functional domains.

Testing the BLAST+ tools

Let's use `blastp` to query the NCBI BLAST servers for proteins with sequence homology to Rme1.

```
$ cd ncbi-blast-2.2.25+/bin
$ ./blastp -db refseq_protein -evalue 2e-10 -remote < ~/tmp/Rme1.fas > ~/tmp/rme1-blast.
    out
```

This runs a remote BLAST search with an E-value cutoff of 2^{-10}. It might take a little while because it will access the NCBI servers to run the search. Note the use of both an input redirection operator as well as an output redirection operator. If you want the output in HTML format for easy reading run it as follows and open the resulting file `rme1-blastout.html` in your web browser:

```
$ ./blastp -db refseq_protein -evalue 2e-10 -remote -html \
        < ~/tmp/Rme1.fas > ~/tmp/rme1-blast.html
```

If all of the commands above worked (produced no errors, generated valid output) then you're ready to move on to implementing the pipelines.

The Pipelines

In today's exercises we're going to look at two very simple pipelines. In next week's class we'll combine and expand on these pipelines using Python.

Pipeline I

This first pipeline will do the following:

1. Read in a protein sequence from a FASTA file
2. Query the NCBI BLAST servers for potential orthologues
3. Analyze the query protein for known protein domains using HMMER and Pfam
4. Prepare a report combining the HMMER/Pfam output with the BLAST output

Here's a simple bash script that implements this pipeline. Save this script as `pipeline1.sh`.

```
#!/bin/bash

# change these if these files are located somewhere else
BLASTP=$HOME/tmp/ncbi-blast-2.2.25+/bin/blastp
HMMSCAN=/usr/local/bin/hmmscan  # change as appropriate
PFAMDB=$HOME/tmp/Pfam-A.hmm

progname="$0" # the name of the program, we won't use this for anything
fastafile="$1" # the input fasta file
outfile="$2" # the output file for the report
```

```
# run blastp
echo "Running blastp"
"$BLASTP" -db refseq_protein -evalue 2e-10 -remote < "$fastafile" > "$outfile"

# run hmmscan and append to outfile
echo "Running hmmscan"
"$HMMSCAN" "$PFAMDB" "$fastafile" >> "$outfile"
```

As we've seen before, the 'she-bang' line simply instructs the operating system to run the script with bash. The next three lines specify where on your system you've installed the `blastp` and `hmmscan` binaries and the Pfam database. This script expects two input arguments – the first of which specifies the input fasta file and the second giving the name of the file in which to create the report. We save these values as the variables `fastafile` and `outfile`. Here the syntax `$1` and `$2` refers to the first and second arguments given at the command line. Once all the variables are defined the script simply runs a remote `blastp` search and `hmmscan` using the given fasta input.

Once you've entered and saved the code for `pipeline1.sh` you need to make it executable:

```
$ chmod +x pipeline1.sh
```

You can then execute the code as follows:

```
$ ./pipeline1.sh Rme1.fas Rme1-report.out
```

Pipeline II

Our second pipeline will do the following:

1. Translate a nucleotide FASTA file to amino acid sequences
2. Perform a multiple alignment of the amino acid sequences using MAFFT
3. Place the translated output and multiple sequence alignments in separate files
4. Run HMMMER on the the translated output and produce a report file.

A small Python script

First we're going to write a Python script to take care of step 2, translating the nucleotide sequence to a protein sequence. Save the following code as `aatranslate.py`.

```
#!/usr/bin/env python

import sys
from Bio import Seq, SeqIO
from Bio.SeqRecord import SeqRecord
from Bio.Data.CodonTable import TranslationError

recs = SeqIO.parse(sys.stdin, "fasta")

for rec in recs:
    try:
        newrec = SeqRecord(rec.seq.translate(), id=rec.id+"_translated",
                                description=rec.description + ' (translated)')
        print newrec.format("fasta")
    except TranslationError:
        print rec.format("fasta")
```

After saving the above code as `aatranslate.py`, make it executable by using `chmod +x`. This script takes a set of FASTA nucleotide sequences from from `stdin` and translates them, sending the output to `stdout`. This small script uses a number of classes and functions from the Biopython library that we'll discuss in greater detail in the next class session. Test your function as shown below. Notice how we use `cat` to pipe the fasta file to `aatranslate.py`.

```
$ less unknowns.fas # confirm that the unknowns are nucleotid seqs, q to quit
$ cat unknowns.fas | ./aatranslate.py | less
```

Combining Python and MAFFT

Save the following code as `pipeline2.sh`:

```
#!/bin/bash

# change these if these files are located somewhere else
TRANSLATE=$HOME/tmp/aatranslate.py
MAFFT=/usr/local/bin/mafft
HMMSCAN=/usr/local/bin/hmmscan
PFAMDB=$HOME/tmp/Pfam-A.hmm

scriptargs="fastafile aafile alignfile reportfile"
E_WRONG_ARGS=85
nexpargs=4
args=$#

# check that we got the expected number of arguments to the script
if [[ $args -ne $nexpargs ]]
then
    echo
    echo "Usage: ‘basename $0‘ $scriptargs"
    echo
    exit $E_WRONG_ARGS
fi

progname="$0" # the name of the program
fastafile="$1" # the input fasta file
aafile="$2" # the output file for the AA translation
alignfile="$3" # the output file for the aligned sequences
reportfile="$4" # output file for the HMMER report

echo "Translating sequences"
cat "$fastafile" | "$TRANSLATE" > "$aafile"

echo "Aligning sequences"
"$MAFFT" --auto --quiet "$aafile" > "$alignfile"

echo "Running hmmscan"
"$HMMSCAN" "$PFAMDB" "$aafile" > "$reportfile"
```

After setting the script to be executable you can run it like so:

```
$ ./pipeline2.sh unknowns.fas unknowns-trans.fas unknowns-align.fas unknowns-report.fas
```

This will produce three files, one containing the translated amino acid sequences, the second giving the multiple alignment of those amino acid sequences, and the third with the PFAM output. Use less or a text editor to take a look at the generated files.

At the beginning of this script we added a little bit of error checking code to insure that the correct number of arguments were provided on the command line. If no arguments or the wrong number of arguments are provided the script gives the user a little usage information and exits gracefully. Try running the script as:

```
 $ ./pipeline2.sh
```

Compare this to what happens when you run `pipeline1.sh` without the arguments:

```
 $ ./pipeline1.sh
```