BCH/BIO 519: LAB EXERCISES/HOMEWORK, Thursday March 5, 2015

Assignments can be started in class today and then finished as homework; **homework is due by midnight next Wednesday (3/11)**. Please be sure to follow the instructions for homework formatting and submission. You should feel free to get advice and assistance from your classmates, but everyone should be completing his/her own (unique) assignment.

Today's exercises will focus on BLAST, in particular on running BLAST locally and on parsing your results using Python. BLAST will run from your CCR directories.

This is probably the longest and most difficult of the homework assignments. But, BioMart is so useful, and BLAST so fundamental…

Note that only Exercises 4 and 5 require programming. So even if you're having trouble with Python, you should be able to do most of the assignment.

Exercise 1: First, you'll get some sequences to use to create a custom BLAST database to use for your searches. For this assignment, you're going to do all your searching using the 1 kb of DNA sequence immediately upstream of all of the genes on mouse chromosome 4. Whoa! you say. Upstream flanking sequence from every gene on the chromosome? There must be hundreds of those! (1161, in fact.)

Lucky for you, there's a very easy way to get these sorts of sequences, using Ensembl's BioMart tool. BioMart is really useful for all sorts of things, and easy to use! (After you do the exercise, play around for a minute or two and check all the different sorts of options.)

1. Open your browser and go to Ensembl (ensembl.org)
2. Click on "BioMart" in the menu bar at the top
3. In the dropdown "choose database" select "ensembl genes 78"
4. In the dropdown "choose dataset" select "mus musculus genes"
5. Click on the "Filters" link on the left
6. In the "Region" pane, check "chromosome" and select "4" from the dropdown
7. In the "Gene" pane, check "Limit to genes…" and select "with MGI IDs" from the dropdown (leaving the "only" button checked as it should be by default)
8. Check "source (transcript)" and select "ensembl_havana"
9. Click on the "Attributes" link on the left
10. Click the "Sequences" radio button
11. In the "Sequences" pane, select "Flank (Gene)"
12. Check "Upstream flank" and type 1000 into the box
13. In the "Header" pane, check the following:
    a. ensembl gene ID
    b. associated gene name

    c. chromosome name
    d. gene start

14. Click on "count" at the top of the page; you should get a count of 1161 out of 43629
15. Click results
16. Choose "export to file" and make sure "fasta" is selected; then click on "go"
17. Voilá! A big multi-fasta formatted sequence file containing the 1 kb of sequences 5′ to every gene on mouse chromosome 4 will download (this might take a few minutes, with everyone hitting the server at once).
18. Upload these sequences to mellencamp. Either use a file transfer program such as Filezilla, or use secure copy from your commandline:

    scp *your_file username@mellencamp.ccr.buffalo.edu:.*

    (make sure you have the period "." following the colon; this specifies the current directory. If you want to copy into a different directory, provide a path following the colon instead of the

[Want to make a quick check that you have 1161 FASTA sequences in your file? Use grep with the "-c" option: grep –c '>' *filename*]

Exercise 2: Next, you will create a custom BLAST database to use for your searches. This is done using the *makeblastdb* program, part of the BLAST+ package. There are a variety of parameters that can be used with *makeblastdb*, but we will need only a few. For full documentation, type "makeblastdb –h" on the command line after loading the BLAST module:

To access the BLAST programs on mellencamp.ccr.buffalo.edu, first enter "module load ncbi/blast-2.2.29" at the command line. After this, you should have access to the entire suite of BLAST programs.

*C*ommand line options you will need for *makeblastdb* are:

-dbtype ['nucl', 'prot'] This specifies whether making a nucleotide or protein database

-in [file_name] Name of the input file/database, default is '-'

-out [string] Name of BLAST database to be created, default is input file name

Using *makeblastdb* and the appropriate command line options, create a BLAST database from the file you downloaded in Exercise 1. **Name your database "*your_last_name*-blastdb".** If you are successful, you will see that the files ***your_last_name*-blastdb.nhr, *your_last_name*-blastdb.nin, and *your_last_name*-blastdb.nsq** have been generated.

Exercise 3: Now, we're going to BLAST the sequence file back against the database. This isn't as silly as it sounds: I've done this type of thing before, e.g. to determine how similar to one another a large collection of sequences is. At any rate, it's good for teaching purposes.

The syntax for running BLAST from the command line depends on which BLAST program you are using (e.g. blastp, blastn, blastx, etc.). You can type 'blastn –help' to get a list of options. Ones you might need for this exercise are:

-query [file_in] These are the sequences you are searching against the BLAST database

-task [String, Permissible values: 'blastn' 'blastn-short' 'dc-megablast' 'megablast' 'rmblastn'] Task to execute; Default = `megablast' (OK to use megablast for this assignment)

-out [file_out] The name of the output file

-evalue [real number] The threshold Expect value; E-values greater than this number are not reported. The default is 10; I usually set it lower.

-db [datbase_file] The database you are searching against. You created a database name earlier when you ran *makeblastdb*. (The database name is the name you used when creating the database, *without any extensions* such as .nhr.)

-word_size [Integer, >=4]   Word size for seeding (length of best perfect match)

-outfmt [string] Determines the output format. "0" corresponds to what you usually see with the online version and is the default; 6, 7 and 10 are tabular formats and can take additional options.

Exercise 3a: Run *blastn* with the following options: database= your database from Exercise 1 (the database name is the name you used when creating the database, *without any extensions* such as .nhr.); input file = your mouse sequences; expect threshold= 1; output file= last_name.ex3a.txt, task = blastn.

This shouldn't take too long to run (but won't be instantaneous). Take a look at the output; it should look a lot like BLAST as you're used to it. Now, we're going to want to do a number of things with this output. We'll want to eliminate uninteresting results, such as the match of each sequence to itself. Maybe we'll want to get rid of anything that doesn't have a certain percent match to the query sequence. Perhaps we want to determine the average number of hits per sequence for our sequences that we just BLASTed.

Think about doing all of these things using the Python you've learned and this output. Now think to yourself: "there's got to be a better way!"

Exercise 3b: Fortunately for you, there *are* other ways. Do this: run the BLAST from exercise 2a again, but this time specify output format '7', "tabular with comment lines". Save this file as "last_name.ex3b.txt".

Exercise 4: Now write a Python program to determine the ***average number of hits per sequence*** using the output from exercise 3b.

There is one wrinkle, however: the tabular format doesn't combine multiple hits to the same sequence into a single summary line the way the default format does—it only outputs the individual alignments. Therefore, when calculating your averages you should only consider one hit per subject sequence. For example, the following should only count as 3 hits:

```
# BLASTN 2.2.17 [Aug-26-2007]
# Query: abd-A_PRE_iab-2|abd-A|3R|size=534|PMID=10644409|type=CRM
# Database: testdb
# Fields: Query id, Subject id, % identity, alignment length, mismatches, gap openings, q. start, q. end, s. start, s. end, e-value, bit score
sequence_A     sequence_A     100.00   534    0    0    1     534    1      534    0.0     1059
sequence_A     sequence_A     100.00   18     0    0    259   276    276    259    0.008   36.2
sequence_A     sequence_B     95.00    20     1    0    13    32     1212   1193   0.12    32.2
sequence_A     sequence_C     100.00   15     0    0    261   275    25     39     0.49    30.2
sequence_A     sequence_C     100.00   15     0    0    260   274    39     25     0.49    30.2
```

Name your program *last_name.ex4.py.* You can just output to *stdout*. Your program should report (1) the number of queries (this ought to be 1161!), (2) the number of hits/query, and (3) the average number of hits/query. When I ran it I got:

```
total hits is 67079.0, total queries is 1161.0
Average hits/query is 57.7769164513
```

Some hints:

The output file is highly structured. When it moves from one query sequence to the next, the set of four header lines are printed over again. You can take advantage of this to note when you are moving to a new query.

Unless you specifiy that numbers are floating point (float(number)), division of two integers will always round to an integer; or remember to import 'division' from 'future'.

One easy way to eliminate duplicate values from a list is to use a dictionary. Since dictionary keys must be unique, if you read your list into a dictionary Python will only store each key once, even if it appears many times in your list. For example, say you read in a file in which each line contains the text "same line" and store these lines in dictionary 'dict'. If you print all the keys in the dictionary, the only thing you will get is "same_line":

```
for key in dict.keys():
        print "The key is {0}" .format(key)
```

will print just the single line "same_line", one time. If your file had an additional line that read "different line", the code block above will now return:

```
The key is same line
The key is different line
```

(If you had made your dictionary using code such as:

```
for line in file:
        line.strip()
        dict[line] += 1
```

then the *value* of dict['same line'] will be 15, and the *value* of dict['different line'] will be 1; but you don't care about the values here.)

If all you want to know if the number of keys in dictionary "*dict*", you can call the *length* function:

```
len(dict)
```

Exercise 5: Now let's make a revised output file that's smaller and easier to read. Name your program *last_name.ex5.py*. Your new output file should contain only the following fields:
Query id, Subject id, % identity, alignment length, e-value
Moreover, you should eliminate

1. hits to self (query=subject)
2. hits with less than 95% identity
3. hits with E-values greater than 0.1

Name this file *last_name.ex5.output.*

In a separate file, named *last_name.ex5.stats* have the program report

1. the total number of hits
2. the number of hits with an alignment length of less than 20

I got:

```
total number of hits is 3161
number of hits less than 20 is 604
```

assuming I did it correctly.

Remember: you will need to convert numbers to floating point numbers to do > and < operations on them. So for example if you have the E-values assigned to a variable "evalue" make sure to convert it to "float(evalue)" before checking if it's > 0.1.

Exercise 6: In Exercise 3 you ran the BLAST specifying 'task = blastn'. The default setting is not blastn but rather 'megablast'. Re-run the BLAST as in Exercise 3b, but this time use the default setting (or 'task = megablast') and rename your output file *"last_name.ex6.blast.txt"*. Now use these results with your program from Exercise 4. You should get a different answer than you did for Ex. 4. Save the results as "*last_name.ex6.txt."*

Exercise 7: Why does 'megablast' give you a different answer than 'blastn'? Run your BLAST one more time, again using 'type = blastn' like you did for Exercise 3b, but add an additional command line option that will cause it to run just like 'megablast' in Exercise 6 (and change the output file name to *"last_name.ex7.blast.txt"*). Check that you did it right by using these results with your Exercise 4 program: you should get the same results now as you did in Exercise 6. Provide the command line instructions you used a file named *last_name.ex7.txt*.

Submitting the homework
Make sure you follow these instructions or your assignment may not be graded!
Homework is due by midnight Wed. 3/12. For this assignment, submit using the command submit --hw=6. Submit each of your program and output files. In addition, like you did for last week, create a terminal session *script*. Name this session *last_name.hw6.txt*. Once this is running, do the following:

- type *cat last_name.ex4.py* to print out your program from exercise 4
- run the program from Exercise 4 (if you directed the output to a file, make sure you then "cat" the file)
- type *cat last_name.ex5.py* to print out your program from exercise 5

- run the program from Exercise 5
- type *head –n 25 last_name.ex5.output* to print just the first 25 lines of the output
- type *cat last_name.ex5.stats* to print the statistics file
- type *cat last_name.ex6.txt* to print your exercise 6 results
- type *cat last_name.ex7.txt* to print your exercise 6 results

NB: As mentioned in class, there are BLAST parsing functions built into BioPython, and lots of others have been written as well.