Programming for Bioinformatics | BIOL7200

Estimating genomic distances using the approximate Jaccard

Do not assume that the user gives you correct inputs all the time. **Your script will be graded on the output produced and how the errors are handled.**

There will be a CI provided during the final week of the assignment. **The public CI will only test core functionality.** You will be expected to test your error handling.

For this assignment, the module list is "`sys`", "`re`", "`os`", "`random`", "`argparse`", "`multiprocessing`", and "`threading`". The grading CI will not install missing modules. Do not use `input()` for any input.

Instructions for submission

- **This assignment is due Tuesday, December 4, 2018 at 11:59pm. Late submissions will not be graded**
- The CI will be made available to you on November 27th to Dec 3rd at 11:59pm
- Your code must be available on GitLab at the above time to be graded
- Name your script as **genomeDist.py**
- Your code should run as `./genomeDist.py <options and inputs described below>`
- **DO NOT HARDCODE** any file name!
- Please use the `#!/usr/bin/env python` as your shebang
- **Your script should finish within 10 minutes**

Estimating genomic distances using the approximate Jaccard

Max score: 100 points

In genomics, quantitatively comparing two or more genomes (or individuals or isolates) is a common task. We do this when we are performing any sort of population genomic analysis such as comparing genomic distances between human individuals (Hamming distance), genomic distances between microbial genomes (ANI, MLST, etc.) or genomic distances between microbial communities (Bray-Curtis, UniFrac, etc.). Next semester in BIOL 7210, you will almost certainly be doing some sort of whole-genome comparison. In the pre-genomics era, genomes were compared using a technique called **DNA-DNA hybridization** (DDH). DDH is a time consuming, costly, and dangerous (radio-labeled DNA) process that does not scale well to tens of samples. A computational approach that can be thought of as *in silico* DDH, Average Nucleotide Identity (ANI), was introduced in 2007 and uses BLASTn or other sequence alignment techniques to estimate a whole-genome alignment by aligning ~1000 bp chunks of two genomes. For those interested in learning more, this page: http://imedea.uib-csic.es/jspecies/about.html contains a good, simplified discussion on what ANI is.

ANI scales to tens and even a few hundred genomes but is computationally expensive. A relative recent technique, Mash, was introduced in 2016 and approximates genomic distance using the MinHash. Mash estimates genomic distances by comparing sets of kmers between genomes and computing an approximate **Jaccard index**.

For this bonus exercise, we want you to implement a simplified version of the core Mash algorithm in Python and return the **Mash distance**, *D*, for two or more input genomes. *This whole problem is algorithmically and conceptually simple so don't get afraid from the set theory formula below. We have* *tried to be explicit about the complete process in this document so you have a clear understanding of the problem statement. The solution, however, is simple.*

1. ***What are you provided with?***
   You are provided with one or more genomes serving as your input. These should be nucleotide FASTA files. You are welcome to reuse code from the kmer couter or all2fasta scripts, try importing them as modules.

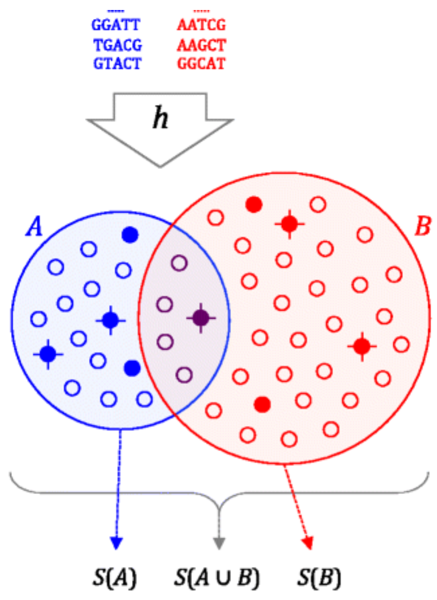
2. ***What is Jaccard (j) and Mash distance (D)?***
   The Jaccard index is a measure of *similarity* between sets and is defined by equation (1):

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

Jaccard distance, a measure of *dissimilarity* between sets, is obtained by subtracting the Jaccard index from 1.

$$d_J(A,B) = 1 - J(A,B) \tag{2}$$

In the case of genomic sequences, your set is comprised of words length k (i.e. kmers) derived from genome sequences. The below image depicts calculating an estimate of Jaccard index from two genomes (A, blue and B, red) using $k = 5$ and a set size, $s = 5$. A random subset (s) is used to simplify the computational requirements.



Modified Figure 1 from Ondov, et al 2016

Mash distance, *D*, is proportionally related to the Jaccard index calculated for a pair of genomes. Mash distance is more robust than a simple Jaccard and, with the appropriate assumptions, can closely approximate ANI-based distance measures with significantly less computation.

$$D = -\frac{1}{k} \ln \frac{2d_J}{1 + d_J}$$

(3)

Where k is the kmer size and d_j is the Jaccard estimate (from equation 2) using a sketch size s.

3. *How do you get your kmer?*
   There is an additional wrinkle in how you should kmerize your genome sequences. For each window, length k, find the ***forward and reverse-complement*** and store the lexicographically smaller of the two. For example, for the 5-mer "ATGGCA" and its reverse complement "TGCCAT", "ATGGCA" is lexicographically smaller. Finding the smaller kmer in Python is very easy, don't over complicate things. The lexicographical order can be found using the sort function.

4. *What are you expected to do?*
   We expect you to implement the below options / flags using argparse

Flag / Option	What it does
`-a`	An input genome
`-b`	Another input genome
`-d | --dir`	Directory containing input genomes files (`.fasta, .fna, .fas`)
`-s | --setsize`	Number of random kmers to evaluates, default to `1000`
`-S | --seed`	Seed value for random set generation
`-o | --output`	Output file name, default to "`genomeDist.txt`" if `-o` is provided with no additional strings. If `-o` is not provided, write to `STDOUT`
`-t | --threads`	Number of theads/processes to run the analysis, defaults to `1`
`-v | --verbose`	Verbose mode
`-f | --force`	Overwrite files if they exist. Do not overwrite by default
`-h | --help`	Help text

Below are listed the types of input your script should accept and what the expected outputs are:

1. If `-a` and `–b` are given, compute the pairwise distance. Output a single, tab-separated line:

```
#a_file_name b_file_name     distance
```

   Please note that the space above are single tab spaces.

2. If `-a` and `–d` are given, compute the all-against-a pairwise distance. Output in two column, tab=separated format:

```
#            a_file_name
Genome_1     distance1
Genome_2     distance2
…            …
GenomeN      distanceN
```

3. If only `-d` is given, compute the pairwise distance and output a square, tab-separated matrix

	Genome1	Genome2	…	GenomeN

Genome1	0	.01	…	.4
Genome2	.01	0		.42
…	…	…	0	…
GenomeN	.4	.42		0

Please note that the above values are only for example purposes.

5. *What errors are you expected to catch?*

   You should not assume that your user has read the documentation, therefore you need to check for the common pitfalls listed below.

   - User supplies -a without -b or -d
   - User supplies -b without -a or with -d
   - User supplies -a, -b and -d
   - User supplies an option which does not exist
   - User supplies an empty directory for -d
   - User supplies files that are not nucleotide FASTA
   - Threads are positive integer values (thread ≤ 0 is obviously senseless)

   You are welcome to catch other types of errors (like no write permission for output) but these are not required. Try-Except and If-Then statements will be your friends for this script.

6. *Where does the parallelization bit comes in?*

Each pair of genomic comparison is independent from each other. This is a great example of an embarrassingly parallel problem. Embarrassingly parallel problems are the easiest problems to implement as each task can be calculated without looking at other running/about to run tasks. Consider this example:

If I need to calculate genomic distance between 4 genomes, I will have to do 4 x 4 pairwise calculations. Given that the distance is non-directional (i.e., distance of genome1 to genome2 is the same as distance of genome 2 to genome1) and distance to self is always 0, we can get away with calculating one half of the matrix (lower triangle or upper triangle). Thus, for 4 genomes, we really need to calculate 6 pairwise distances (generally speaking, for n genomes, we need to calculate n*(n-1)/2 distances). If I run my genomeDist.py with 6 threads, I can launch 6 threads and give one pairwise calculation to each thread, wait for them to finish and generate my matrix. Theoretically, this will give me a 6X speed-up compared to when I run the script on a single thread.

Deliverable: Python script (`genomeDist.py`) which takes on three argument from the command line.

Syntax: `./genomeDist.py –a <fasta> -b <fasta> [-d|--dirs <dir of fastas>] [-o|--output] [-t|--threads] [-v|--verbose] [-h|--help]`

Example usage 1:

```
./genomeDist.py –a genomeA.fasta –b genomeB.fasta –o genomeDist_output.txt –s
100
```

Compares genomeA and genomeB, using 100 random kmers, writes output to "genomeDist_output.txt"

Example usage 2:

```
./genomeDist.py –a genomeA.fasta –d genomesFolder –o a_verse_dir.txt –t 10
```

Compares genomeA and the genomes in directory "genomesFolder", using ten threads and writing the output to "`a_verse_dir.txt`"

Example usage 3:

```
./genomeDist.py –a genomeA.fasta –d otherGenomes –o a_verse_dir.txt –t 10
```

When run immediately after example 2, your program should quit because the output file exists

Example usage 4:

```
./genomeDist.py –a genomeA.fasta –d otherGenomes –o a_verse_dir.txt –t 10 -f
```

Compares genomeA and the genomes in directory "otherGenomes", using ten threads and overwrites the output file, "`a_verse_dir.txt`", with the new data

Again, keep in mind:

- Do not use any Python modules other than `sys`, `re`, `os`, `random`, `argparse`, `multiprocessing`, and `threading`. You may write and import your own modules, just make sure they're on GitLab too.
- Do not hardcode or use input()
- Use shebang line