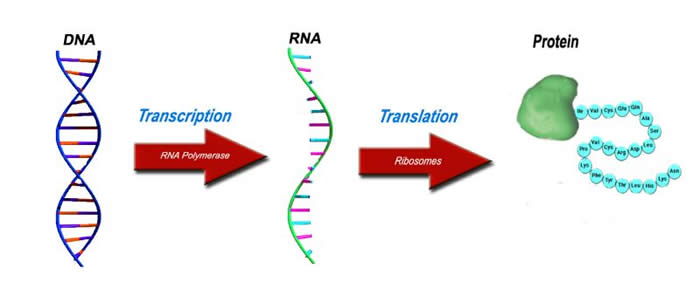
Paul Kadota

5/22/17

Python: A Pipeline for Understanding Genetic Information

Python is a perfect tool for genetic analysis as it incorporates a high-level programming environment with a large library of bioinformatics packages. With the advent of next generation, high-throughput sequencing technology, there is an urgent need for reproducible and scientifically-validated pipelines to convert the copious amounts of genetic ‘data’ into actionable ‘information’. Python excels at this process as the Python community has created and maintained numerous packages that can transform raw DNA sequences into myriad of transformations and analytical tools. In particular, Biopython has many sub-libraries that make processing raw FASTA or FASTQ files much simpler. In this analysis, I used Biopython, as well as, user-defined Python functions to parse, transcribe, translate, and align complex nucleotide sequences into genetic knowledge.

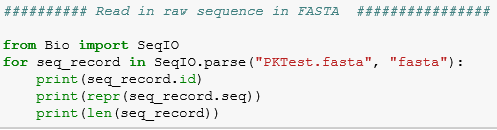
The ‘Central Dogma of Molecular Biology’ (pictured below) is the transfer of inherited genetic information from genotypic DNA to phenotypic proteins. This process starts when the double-stranded DNA strands are copied into RNA in a process known as transcription. Subsequently, the messenger RNA (mRNA) is translated into amino acids. Chains of these amino acids are building blocks to complex proteins that regulate, support and become the building blocks for all living organisms. The ability to understand and manipulate this core function of our genetic make-up is one of the most exciting and important development of our human civilization, and Python is the key to unlocking these fundamental revelations.



http://www.lhsc.on.ca/Patients\_Families\_Visitors/Genetics/Inherited\_Metabolic/Mitochondria/DiseasesattheMolecularLevel.htm

There are many challenges in the realm of genetic analysis; however, Python can overcome some of these hurdles. The first major challenge in parsing information from genetic data is the sheer size and complexity of the genome. The human genome is thought to have 3 billion nucleotides comprising over 22,000 individual genes. In the initial stages of The Human Genome Project, this unwieldy volume of data took the best minds hundreds of thousands of man-hours over an entire decade while spending over a billion dollars to sequence just one human genome. Luckily, computer programming languages, especially Python have reduced the overwhelming task of analyzing billions of nucleotides by automating several key steps. Computer programming languages are intrinsically an ideal tool to create genetic pipelines. The computer language is already set up to read and manipulate long string of data. DNA is essential one long string of letters that define living world. The high-level programming environment of Python allows us to probe and understand this magnitude of genetic information. On top of the excellent programming environment, one of Python’s greatest strengths is the community of fellow Python programmers that create and maintain a host of bioinformatics packages. These libraries, including Biopython, hold virtually innumerable pre-made batches of code that can create reproducible genetic pipelines.

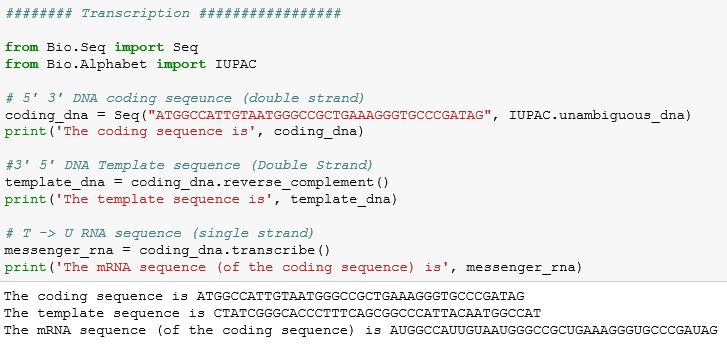
The first step of any genetic pipeline is the procurement of raw data. As the cost of sequencing has plummeted (far surpassing Moore’s Law), the availability of whole genomes has become widespread. The majority of raw DNA output files are in FASTA or FASTQ format. FASTA includes a unique identifier along with the nucleotide sequence; FASTQ has contains an additional line of quality scores that give a rating for the confidence of the identity of each nucleotide. For my conceptual pipeline, I used FASTA files as my raw data. I initially parsed out just the DNA sequences with Biopython (below).





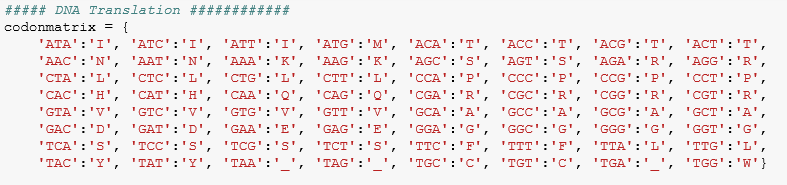
The package, Biopython, has a very easy to comprehend sub-library, SeqIO, to handle the input/output of DNA/amino acid sequences.

The next step of my genetic pipeline was to demonstrate the aforementioned ‘Central Dogma of Molecular Biology’, and transcribe the DNA to RNA, and then translate the RNA to amino acids. Again, pre-made libraries in Biopython made this step easy to accomplish. The biology of transcription incorporates special enzymes that cleave apart the double-stranded DNA so that it can be copied to single-stranded RNA, specifically mRNA. In the cell, extra consideration must be made to copy the ‘template’ (3’-5’) DNA strand as directionality is important. In silico, however, the math works out such that simply replacing thymine (T) of the ‘coding’ strand of DNA with the appropriate uracil (U) of RNA creates the correct sequence (below).

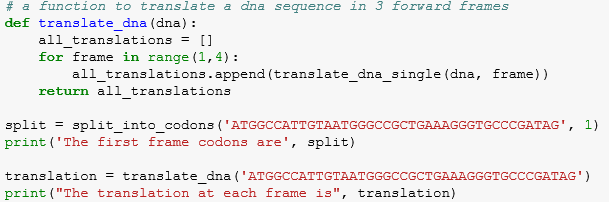


This final mRNA sequence is now ready to be translated.

Translation is the second step of converting RNA to amino acids. In this step, I created a matrix of codons that represent every triplet combination of nucleotides (below).



These codons are the building blocks to amino acids. Each triplet encodes for a specific amino acid. As one could start ‘reading’ each line of DNA from either the very start of the line, one step in or two steps in, three different ‘reading frames’ must be considered. The product chains of amino acids will be very different depending where you start reading in the data. A small cohort of ‘stop’ codons (TAG, TAA,TGA) terminate the translation process. At the end of this step, the output will be chains of amino acids that may become function proteins (below).





From here, the next step is to determine the identity of genetic information by performing sequence alignment. By comparing a sample DNA string to a known DNA sequence, one can postulate the identity of the queries sequence. In Python this sequence alignment can be accomplished in many ways. In order to better understand this process, I chose to avoid alignment tools that contained a ‘black box’ of unknown analysis, and I used Python functions to perform sequence alignment. The first step of this process was to create a scoring matrix for each potential match. The most basic algorithm was a ‘+1’ for an exact match and a ‘-1’ for any mismatch (below).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scoring Matrix** | **A** | **T** | **G** | **C** |
| A | 1 | -1 | -1 | -1 |
| T | -1 | 1 | -1 | -1 |
| G | -1 | -1 | 1 | -1 |
| C | -1 | -1 | -1 | 1 |

Using a ‘brute force’ method of testing every possible combination of alignments, I was able to find the top 10 best matches. Although I did not use it, another very popular alignment protocol is Basic Local Alignment Search Tool (BLAST). BLAST is much more refined than my method as it ‘pre-seeds’ the sample sequence with areas of high identity compared to the reference sequence. In this way, BLAST is an efficient and ideal method of sequence alignment. As such, there is a Python-based build for BLAST in Biopython.

The last part of my analysis was to use sequence alignment in a slightly different way to find the biological homology of a sequence of genetic information between human and mouse. I used another Biopython library, ‘pairwise2’ to run another alignment of a specific oncogene, p53. Despite being from different organism, both species showed conservation of genetic sequence for p53.

Through the application of Python, I was able to create a working pipeline of genetic analysis tools that read in raw data and ultimately transformed these jumbled nucleotides into known sequence of amino acids. The available libraries of Python greatly aided my endeavor as they allowed me to quickly move in and around the data without having to ‘reinvent the wheel’. Moreover, the scripts ran incredibly efficiently on my personal computer. This ability to parse out genetic information from our DNA shows the power of Python programming.