Numbat K-mer Analysis Tool

Version 0.1

Shawn.M.Rupp@asu.edu

Contents

Introduction

Numbat is a series of scripts for analyzing k-mers in NCBI flat files and comparing codon frequencies reference data set.

Dependencies:
Python3
Cython
Kestrel

Installation

Cython
Several of the scripts utilize Cython, which compiles python code into C and drastically improves performance. Cython can be installed from the pypi repository or via Miniconda (it is installed by default with the full Anaconda package).

> To install with Miniconda:
> `conda install cython`

Numbat
Download the repository:

```
git clone https://github.com/icwells/Numbat.git
```

Most of the scripts are written in python3, but several contain Cython modules which must be compiled. Running the install script will compile the Cython scripts and move the binaries to the bin directory:

Change into the Numbat head directory and run the install script:

```
cd Numbat/
./install.sh
```

Kestrel
If you wish to compare codon frequencies with a reference data set, you must download the Kestrel Taxonomy Tool and install it (be sure to check its read me for information about getting source web page API keys).

```
git clone https://github.com/icwells/Kestrel.git
cd Kestrel/
./install.sh
```

Getting Started

Extracting Sequences from Flat Files
Presuming that you are starting with a flat file of reference sequences, the first step is to extract coding regions from the file. This can be done with:

```
cd bin/
extractORFs.py –i <path to flat file>
```

This will create a fasta file with one gene per line and a header with the following format:

>Accesion-ProteinID-ProteinName-[coordinates]__Host

This file can be used for input to kCLuster.py and cladeFinder.py in no particular order.

Local Alignment via K-mer Clustering
kCLuster will use a machine learning algorithm called k-means clustering to sort reference sequences into groups of closely related sequences. Closely related sequences are defined as those with the most similar length and number of unique k-mers (as a proxy for overall diversity). A separate fasta file of gene sequences can then be aligned against the clustered sequences. Each input gene will be assigned to the group of reference genes it is most similar to and will then be locally aligned to each gene in the cluster.

To cluster reference sequences, run the following:

```
python kCluster.py ––cluster –k <k–mer length> –n <# of genes
per cluster> –i <path to fasta input> –o <path to output
directory>
```

The k-mer length must be a multiple of 3 in order to iterate over the sequences by codon. The -n option determines how many clusters there will be. Since that is not very informative in terms of running the program, the number of clusters is determined by dividing the total number of genes by n to give a rough idea of how many genes are in each cluster (the exact number for each cluster depends on how the genes are sorted). Lastly, the -o option points to the directory where the clustered sequences will be written. Be sure this is either an empty directory or a new directory (kCluster will make the directory if it does not exist) since the program will write one file per cluster to ease the memory load later on. This can take several hours, depending on the input and the number of clusters (more clusters means more calculations).

To align gene sequences, enter the following:

```
python kCluster.py –t <# of threads> –a <significance threshold>
–m <path to clustered sequence directory> –i <path to fasta
input> –o <path to output file>
```

kCluster is capable of multiprocessing, and one input gene will be aligned per thread given by the -t flag. The -m flag points to the directory made in the previous step, while -i indicates a fasta file of query sequences. The -o option indicates the output file, which will be in csv format and have the following columns:

Query	The name of the query sequence
k-merLength	The number of k-mers used for clustering reference sequences
queryLength	The length of the query sequence (in nucleotides)
Hit	The name of the aligned reference sequence
HitLength	The number of aligned k-mers
PercentIdentical	The number of identical matches between the query and the hit
E-Value	The probability of getting a random match of the given length and quality

Since the k-mer length is in nucleotides and the hit length is in k-mers, it is possible to have a 100% hit the is up to k nucleotides shorter than the input sequences. The output has been left this way to keep it as descriptive as possible.

K-mers are considered identical if they have the same sequence as the k-mer with the same index in the aligned sequence. Therefore, only k-mers that match “as-is” in each sequence are considered identical.

The e-value is calculated with the following formula:

$$e = (\text{total}/4^k)/(\text{hit length}*\text{score})$$

where total is the total number of nucleotides in the reference clusters, k is the k-mer length, hit length is the number of aligned k-mers, and the score is an internal percent score similar to percent identical but accounting for insertions and deletions. In simpler terms, e is equal to the chance of of any random k-mer from the total reference input ($\text{total}/4^k$) equaling a given k-mer. This figure is then divided by the length of the hit (the number of times it is repeated) multiplied by the percent score to account for any lack of identical sequences in the alignment.

The -a flag specifies the significance value (i.e. maximum e-value to report). This is set to 0.05 by default, so any e-value above 0.05 will not be written to the output file.

Identifying Hosts via Codon Frequencies

cladeFinder will take the same input file as kCluster and calculate codon frequencies for the reference sequences. While comparing codon frequencies is easy enough, resolving host names withing the flat file and getting their taxonomy is not, so you must first extract the host names from the fasta sequence and use Kestrel to find taxonomy information.

To extract host names:

```
python cladeFinder.py --extract -i <path to input fasta> -o
<path to output file>
```

The output file is a text file with one host name per line. This can be used as input for Kestrel. To call Kestrel:

```
python cladeFinder.py -t <# of threads> -c 0 -i <path to host
text file> -o <path to output file>
```

The output file will be a csv file containing the host name, downloaded taxonomy, and source information. The -c flag indicates the column number of the host name, which is 0 in this case since there is only one column in the input file.

Once Kestrel has finished, the reference codon frequencies can be calculated at either the family or the genus level (species level quantification would be too specific). Running the following:

```
python cladeFinder.py --quantify -r <path to Kestrel output>
-i <path to input fasta> -o <path to output file>
```

will produce a csv file with one entry per family (or genus if –genus is given) of hosts in the reference fasta. Each line will have the proportion of the frequency of each possible codon found in that clade.

After reference frequencies have been calculated, genes from unknown hosts can be compared against the output.

```
python cladeFinder.py --quantify -m <maximum difference> -r
<path to quantified frequencies> -i <path to input fasta>
-o <path to output file>
```

This will calculate the mean of the absolute value of the difference in the frequency of each codon. If the average is less than the maximum given with the -m flag, the hit is written to the output file. This analysis is extremely sensitive at this point, so the changing the maximum difference can have a substantial impact on the number of results written.

The output csv has columns for the query name, hit name, and the average difference. Unfortunately, there is no way (as of yet) to get a significance value for this comparison, so the output should be verified if possible.

Scripts

extractORFs.py
This script will extract open reading frames from NCBI flat files in fasta format. Output will be written to the same directory as the input file.

arguments:

```
-h, --help  show this help message and exit
-v          Prints version info and exits.
-i I        Path to input flat file.
```

kCluster.py
kCluster will perform k-means clustering on a fasta file of reference sequences and perform a k-mer based local alignment between a fasta of input reads and clustered reference sequences.

arguments:

```
-h, --help  show this help message and exit
-v          Prints version info and exits.
--cluster   Quantifies k-mers in reference genes.
-k K        Length of k-mers (default = 15; must be a multiple of
             three).
-n N        Approximate number of genes per cluster (default =
             1000).
-m M        Path to trained cluster directory.
-i I        Path to input fasta file.
-o O        Path to output file/directory.
-a A        Alpha value for e-value (default = 0.05).
-t T        Number of processors to use for analysis (default = 1).
```

cladeFinder.py
This script will quantify the frequencies of codons in gene sequences and compare between reference and target sequences.

arguments:

```
  -h, --help  show this help message and exit
  -v          Prints version info and exits.
  --extract   Extracts unique host names from reference fasta created
               with extractORFs.py.
  --quantify  Quantifies codons in reference genes.
  --genus     Quantify codon frequencies at the genus level
               (quantifies at family level by default).
  -r R        Path to reference taxonomy file/quantified codon
               frequencies file.
  -i I        Path to input fasta file.
  -o O        Path to output file.
  -m M        Maximum allowable difference between query and hit
               frequencies (default = 0.1).
```