Draft K-mer processing plan

Update by D. Maier, 11 July 2016

This is the initial plan for generating and using K-mers from the Chisholm Lab sequence data.

Global Variables

`K`, the length of the desired K-mers.
`KMers`, a list of all `KMers` of length K.

Utility functions

`RevComp(Seq)` gets the reverse complement of a sequence. For example:

```
RecComp(‘ATTGCTT’) = AAGCAAT
```

`Digest(Len, Seq)` is the set of contiguous subsequences of length `Len` of sequence `Seq`.

```
Digest(5, ‘ATTGCTT’) = {‘ATTGC’, ‘TTGCT’, ‘TGCTT’)
```

Abundance Matrix Construction

Step 0: Preprocessing (which we might try to replicate or include one day). Sequences are grouped by sample, trimmed, quality controlled and overlapped when possible. For each sample, there are two files, one for overlapped reads and one for non-overlapped reads. Example file names for sample S0445

S0445_**overlapped_qctrimmed_pairs**.fastq
S0445_**nonoverlapped_qctrimmed_pairs**.fastq

Utility functions that construct the file names for a sample.

```
OverlappedFile(SampleNo)
NonoverlappedFile(SampleNo)
```

Step 1: Ingest. Assume we have a list `Samps` of samples we want to include. We want to populate two tables

OLReads(SampleNo, SequenceID, JointSequence)
NOLReads(SampleNo, SequenceID, Strand, Sequence)

Here `SequenceID` is the sequence ID from the sequencing process. `Strand` is 1 or 2, to represent which end the sequence is coming from. It is the last digit of the `SequenceID` for non-overlapped samples.

Step 2: Digest. (Ideally, we can pipeline this step with the next.) We keep the overlapped and non-overlapped reads separate at this point.

```
OLKmers = unnest({(Samp,
 flattenMap({Digest(K,OLReads[Samp, SeqID].JointSequence)})
            |Samp ∈ Samps})
```

`OLKmers` will be a list of `(SampleID, Kmer)` pairs.

```
NOLKmers =unnest({(Samp,
 flattenMap(union(
  {Digest(K,
   NOLReads[Samp, SeqID, 1].Sequence),
  Digest(K,
   RevComp(NOLReads[Samp, SeqID, 2].Sequence))
  })
            |Samp ∈ Samps})
```

The output schema is the same as for `OLKmers`. Note that we reverse complement the second read of the pair so all sequences are read from the first strand.

Discussion: Feedback from Steve Biller is that direction shouldn't make a difference, but that it might be good to add in each k-mer directly and also for its reverse complement. I would kind of like to keep it this way initially, and do the sanity checks. We could then reprocess the abundance matrix to add reverse-complement counts to every k-mer. If we do go that way, Vaughn Iverson points out we can eliminate half the k-mers by only storing for a k-mer string if it's lexicographically less than its reverse complement. For example, store for ACG but not CGT, since ACG < CGT.

Step 3. Count. Create the abundance matrices.

```
OLKmerCounts =
 select SampleID, Kmer, count(*)
 from OLKmers
 group by SampleID, Kmer
```

We want to turn this into an array
`OLKmerFreq(frequency: Int)[Samps, Kmers]` with default value 0.

Similarly

```
NOLKmerCounts =
 select SampleID, Kmer, count(*)
 from NOLKmers
 group by SampleID, Kmer
```

and is converted to the array
`NOLKmerFreq(frequency: Int)[Samps, Kmers]` with default value 0.

We can combined the arrays if needed:

```
AllKmerFreq[Samps, Kners] = OLKmerFreq + NOLKmerFreq
```

Sanity Checks

Once we have these arrays, we can do some tests to see if the data is plausible. It will be useful to have a distance function to compare the frequency vectors for two samples.

An initial candidate is Bray-Curtis dissimilarity:
https://en.wikipedia.org/wiki/Bray%E2%80%93Curtis_dissimilarity
Let `FreqVect1` and `FreqVect2` be of type

```
  Array(frequency: Int)[Kmers]
```

then

```
BrayCurtisDissim(FreqVect1, FreqVect2)  =
1 - 2*sum(min(FreqVect1, FreqVect2)/
       (sum(FreqVect1) + sum(FreqVect2))
```

Here are a few tests.

Test 1: Reverse-complement comparison. The frequency of a K-mer and its reverse complement should be roughly the same in each sample.

Define `RevComVec(FreqVec)` to be the vector `RCFreqVec` such that `RCFreqVec[km] = FreqVec[RevComp(km)].`

Then we can compute a vector `OLRevCompDists` of comparisons by sample of type `Array(bcDis: Float)[Samps] where`

```
RevCompDists[s] =
  BrayCurtisDissim(
      OLKmerFreq[s, *],
      RevComVec(OLKmerFreq[s, *])
     )
```

Do this also for NOLKmerFreq.

Test 2. OL-NOL comparison. Do we see the same patterns between the OL K-mers and the NOL K-mers?

```
OL_NOLDists[s] =
  BrayCurtisDissim(
      OLKmerFreq[s, *],
      NOLKmerFreq[s, *]
     )
```

This one needs work. We should probably compare relative abundances, since there might be, say, more OL sequences in general than NOL sequences. We can normalize rows in the matrix by total row counts below.

DataSet Summaries

We need some ways to summarize the datasets to get an overall view. We can do some row (sample) and column (k-mer) summaries.

```
SampleSummary[s] =
   (total = sum(AllKmerFreq[s, *],
    avg = avg(AllKmerFreq[s, *],
    spread = max(AllKmerFreq[s, *]
    non-zero = count(AllKmerFreq[s, *] != 0)
```

We could also do a simple histogram of the values in each sample vector.

```
KmerSummary[km] =
  (total = sum(AllKmerFreq[*, km],
   avg = avg(AllKmerFreq[*, km],
   spread = max(AllKmerFreq[*, km])
   non-zero = count(AllKmerFreq[*, km] != 0)
```

`KmerSummary` is too big to view, so it might be further summarized with max, min, avg, top-n, histogram, etc.

We can also compute an all-pairs distance matrix `PairsDists` across samples of type `Array(bcdis: Float)[Samps, Samps]` as

```
PairsDists[s1, s2] =
  BrayCurtisDissim(AllKmerFreq[s1, *] , AllKmerFreq[s2, *])
```

Lookups

The Demo needs a way to poke at a sample (perhaps by clicking it on a map) and get some kind of summary of K-mer information for the sample. One possibility might be to just give the appropriate entry of `SampleSummary`, or show a histogram of the frequency distribution.

Analyses

One kind of analysis will try to make sense of the distance matrix, by multidimensional scaling, say.

Bill Howe suggests the option of t-SNE (https://lvdmaaten.github.io/tsne/) which he says has worked well for him. It appears that t-SNE needs similarities rather than differences. In that case, we might still use the pairwise differences array to get the Sørensen similarity index, which is conveniently defined as

```
   Sørensen = 1 - (Bray-Curtis)
```

We also want to look at analyses that consider the GeoTraces data. For example, for a particular K-mer km we could look at how its frequency varies with a particular physical variable, say via a scatterplot:

```
ScatterPlot(AllKmerFreq[*, km], GeoTraces.depth)
```

Here we're treating GeoTraces as an array indexed on `Samps`.