David Evarts

Demultiplexing and Index Swapping Assignment – Part 2 of Part 1. Writing Pseudo-code for demultiplexing. Due 8/11/19

From the Assignment:
"Goals: Our goal is to look through a lane of sequencing generated from the 2017 BGMP cohort's library preps and determine the level of index swapping and undetermined index-pairs, before and after quality filtering of index reads. In order to do this, we must first demultiplex the data. Develop a strategy to de-multiplex samples to create 48 FASTQ files that contain acceptable index pairs (read1 and read2 for 24 different index pairs), two FASTQ files with index-hopped reads-pairs, and two FASTQ files undetermined (non-matching or low quality) index-pairs."

Set up steps

1.) - Create a list with the 24 index identifiers and their associated barcodes sequences. This list will be a dictionary that associates each bar-code sequence with it's identifying tag (in example B1 goes with GTAGCGTA). I'll name it "**assigned_indices."**
No unit test needed here. I'll just use a print statement to check to be sure this matches the given list of multiplex indices.

2.)- Using arg parse create an input list in which the user inputs four files and identifies which are the first and second index reads and which are the first and second sequence reads. Arg parse should set a flag for each of these files. These will allow us to later refer to each file.

The unit test should be a set of four fastq file names. I'll place a print statement with the arg parse flags identifying then as Index 1, Index 2, read 1 and read 2 behind the arg parse input commands and run it to see if it correctly identifies the files with the type that are user inputted.

3.) **Function – index_barcode**
This function will provide us with indexed lists of lists. There will be one list within a larger list that associates the base with its numbered position in the bar code.

Steps for index_barcode

a.) - Open the index-bar code list.
b.) - Set up a counter.

c.) - Iterate through the list. For each bar code, create a dictionary in the list, which associates the letter with a number from the count, which will be 1-24

Unit Test
I've a tester index file that I created from the first 12 reads of the 2017 fastq file. I can input them and add a print command for the dictionary for the script. I should get a list of dictionary objects, each with

a barcode, as a keys and one of 24 numbers, as a value.

4.) **Function – complement_list**

This function will find the complement to each given index bar code and add them to a dictionary matched to the same number that their reverse is matched to in the index_barcode function.

Steps for complement_list

a.) Open the index bar-code list.

b.) Set up a new dictionary matched pair list named "barcode_complements".

c.) For each barcode sequence, do the following.

i.) Make a complementary code.
ii.) If the first letter in the index sequence is "C" it will be matched to a "G". A "G" will be matched to a "C". An "A" will be matched to a "T" and a "T" will be matched to an "A". The resulting complementary code will be added to the dictionary with the index code as it's key.

d.) Add the complementary barcode with the number to which the original matching barcode is indexed in the index_barcode dictionary into the barcode_complements list.

Unit Test – Using the same index tester. Print the matching barcodes dictionary and the index_barcode dictionary. Check each barcode to see that it is matched to a unique number 1-24. Check that the complement is matched to the same number.

Detecting Possible Index-Swapping and undetermined Reads to Files

This can all be thought of as one large function with sub-functions. This is designed to detect index-swapping and reads with one or both indices that do not match the assigned barcodes. While doing so it labels all of the reads. As the header for each read carries a label with its two indices, the reader of the fie can determine if it is a clear index swapor a read with an index with an undetermined error.

Steps to check the forward index

1.) Using the arg parse flags, open the file for the first read (forward read) set of indexes and the file for the second set of indices.

2.) Iterate over the first index file list and the second index file

3.) Set a counter.

4.) Loop through the sequence for each index in each read, using a counter and a modulo to find the barcode sequence line.

Unit test- Print the index bar code sequences from the tester file.

Steps for checking the second index list

5.) Open the first and second sequence files as well.

6.) Check the second index list for that read using the counter and a modulo to find the barcode sequence line.

7.) Call the **complement_list** function.
8.) Use the index_pair labeler function to relabel the reads.

Function – index_pair_labeller

This function alters the header of a read to display the sequences of the indices associated with it. An example of an edited header for a sequence fie of a read with an index swap would be one in which the first index read AGAGTCCA, but the second index read TCCTATCG, rather than TCTCAGGT. In this case the header would labelled "@AGAGTCCA- TCCTATCG" If it were a correct dual matching index, the header would now read "@AGAGTCCA- TCTCAGGT". The rest of the information in the header would be unchanged.

Steps for index_pair_labeller.

a.) Create another line counter. Use it and the modulo to identify the header line in the current Read.

b.) Edit the header line to start with "@"<the sequence for this read in the index file> and<the sequence for this read in the second index file>, using reg-ex commands and a pipeline.

Steps for checking the second index list and to assign reads with indices that do not match each other or do not match the assigned indices to a file.

8.) Check the second index list to see if the second index for this read matches a complement_list entry.

9.) Check to see if the paired numerical value of the complementary sequence in the complement_list is the same as that of the assigned index in the paired with the same number as the first index for this read in the index_barcode dictionary.

10.) Check the first index against the assigned_indices list.
11.) If the two indices do not match each other do the following.

a.) Check to see if there is a file named with the first and second indices that are at this read and the following "suspected_index_hop_first_reads.fastq." If there is not such a file, create one.

b.) If there is not such a file, also create one named "suspected_index_hop_2nd_reads.fastq."

c.) Copy the full first sequence read (all four lines) into suspected_index_hop_first_reads.fastq."

As you do use the index_pair_label function, involving reg-ex commands and a pipeline to alter the header.

d.) Copy the second full sequence read into "suspected_index_hop_2nd_reads.fastq, while also using the index_pair_label function.

e.) If there is not yet such files, create files named "undetermined 1st reads.fastq" and "undetermined 2nd reads.fastq"

f.) If either index does not match anything in the assigned_indices (for 1st reads) or complement_list (for second reads) move all four lines for the first read into the "undetermined 1st reads" fastq file, using the index_pair_labeller to tag the header as it is moved and second reads into the "undetermined 2nd reads" file, also using the index_pair labeler.

Detecting Low Quality Index bar-code sequences

This could be a function containing a sub-function. It is a loop within the loop above. As, such I'll call it ***index_quality_judge****. I'll set the minimum index quality to 35 but, I'll ask the provider of any dual-paired index read fastq file sets to determine what is a minimum index quality for them, using an arg parse flag and let them know that reads with low index quality can be found in the undetermined files. If they'd like to find theirs, they can be detected by header labels which display their index sequence and it's matching complementary sequence.*

Steps for index_quality_judge

1.) Set the quality threshold to the flag supplied by the user to indicate their preference.

2.) Set a counter. This should provide a given number for each index sequence, that matches.

3.) Open the file for the first set of indices and the file for the first set of sequences.

4.) Loop through the quality codes for each index in each read, using a counter and a modulo to find the quality code line.

a.) Within each quality code line iterate over each of the eight characters.

i.) For each character, run the **convert_phred** function.

-Convert_phred takes each quality code character and using the ord command converts it to a number. We then subtract the required amount to have a numerical Illumina quality code

5.) The converted quality scores are averaged at each nucleotide using the **mean_quality_score** function, written for part 1 of this part 1 demultiplexing assignment.

6.) If the mean quality score for one of the eight characters in the quality code line for an index has a numerical value below the thresh hold, the associated sequence read is moved to the Undetermined 1st reads file.

7.) Repeat all for the second read sets file.

Assigning the remaining reads to their users.

This section can be seen as the end of the loop for each iteration in the "Detecting index swaps and…." Section. Here we will finish demultiplexing, as we have for a read decided wether it should be set aside as index swapped or undetermined.

12.) Do the following for any file that has passed the matching index test and the quality test. And has not yet been assigned to a file.

a.) If there is not yet a file with a name matching the assigned bar code index and its identifier and which sequence read, make one. An example of the file name is "B1-GTAGCGTA-1st-read.fastq" or "B1-CATCGCAT-2nd-read.fastq".

b.) Place all four lines of the read into the appropriate file, while labelling it with the index_pair_labeller function.