Bioinformatics and Data analysis

Software used

List of software used... (html link if no citation available)

FastQC v0.11.7
bbmap v38.05
STAR-2.5.2b
bowtie2 v2.2.9
magicblast v1.3.0
SPAdes v3.13.1
ncbi-blast v2.7.1
SeqKit v0.11.0
Samtools v1.8
minimap2 v2.1 https://github.com/lh3/minimap2
sdust Release 0.1-r2 https://github.com/lh3/sdust
LAST http://home.cc.umanitoba.ca/ psgendb/doc/last/last.html

In Python v3.7.1

1. pandas v0.25.1
2. seaborn v0.9.0
3. matplotlib v2.2.3
4. matplotlib-venn v0.11.5
5. statsmodels v0.12.0

In R v4.05

1. polyester v1.26.0

Mystery Miner is available at is available at...

https://github.com/senorelegans/MysteryMiner

Layout of analysis

This supplemental will follow closely along with the nextflow pipeline, subsequent filtering scripts, and the github README. Inside of the main nextflow folder we have added a folder called RunFirst which should help building the various blast databases and indexes. All custom python scripts used in the pipeline are located under bin/src. Jupyter notebooks (.ipynb) in the src folder were used to generate python scripts with the correspond name (example: script1.ipynb makes nb_script1.py) using notebook2script.py. This provides a nice test driven environment to develop in jupyter notebooks and is inspired by the fastai library https://www.fast.ai/

For the explanations below we are assuming the nextflow out folder is called NF_OUT

1 Nextflow Pipeline

FastQC and Trimming

Reads were verified for quality using FastQC, over-represented sequences and adapter contamination were trimmed using bbduk and then rechecked with FastQC. The adapter file (adapters.fa) is located in the nextflow bin

```
bbduk.sh
in=R1.fastq.gz
in2=R2.fastq.gz
out=R1.trim.fastq.gz
out2=R2.trim.fastq.gz
ref=bbmap_adapters
ktrim=r qtrim=10 k=23 mink=11 hdist=1
nullifybrokenquality=t
maq=10 minlen=25
tpe tbo
literal=AAAAAAAAAAAAAAAAAAAAAAA
```

Unmapped reads from target organisms (Human,Mice,Rat, etc)

Reads were aligned against the target organism using the following commands

```
STAR –genomeDir genome
–readFilesIn R1.trim.fastq.gz R2.trim.fastq.gz
–readFilesCommand zcat
–runThreadN 20
–runMode alignReads
–outReadsUnmapped Fastx
–outSAMattributes All
–outSAMtype SAM
–outFileNamePrefix name
–quantMode GeneCounts
–outFilterMultimapNmax 100
```

```
bowtie2
-q
-p 20
-x bowtie2_index
-1 Unmapped.out.mate1
-2 Unmapped.out.mate2
-S bowtie2.sam
–un-conc unmapped.fastq
```

```
magicblast
-query unmapped.1.fastq
-query_mate unmapped.2.fastq
-db magicblastDB
-infmt fastq
-outfmt tabular
-num_threads 20
-no_unaligned
-out blast.tsv
```

After magicblast, seqkit is used to remove reads that match hits from magicblast. Additionally we have taken 400 reads out before magicblast, and add them back in after so that the assembly with spades won't fail.

Assembly

At this point we have unmapped reads ready for assembly. We assemble the samples as seperate single files, as well as concatenated by each condition (Control, Treatment1, Treatment2, etc), and all samples concatenated together (all). This gives us the different heirarchies of single, group, all. The concatenation step is important for the preprocessing for the LastALL step. It should be noted that spades can fail if you do not set the read orientation properly (=or in the nextflow pipeline). It might also fail if you put in too large of a kmer size (=kmer_size in the nextflow pipeline). A general rule is to go a little under half your read length and set it to an odd number (Example: 75bp reads set kmer_size to 35).

```
spades
–rna
–pe1-or
-k kmer_size
–pe1-1 R1
–pe1-2 R2
-o name_spades
mv name_spades/transcripts.fasta name_spades.fasta
```

Nucelotide Blast

At this point we have contigs ready for nucleotide blast. This will give us taxid numbers we can query jgi with. We also add in fake/dummy contigs at this point as well so downstream processes won't fail. These sequences are located in dummysequences.fa in the bin. This will not be counted in the dataframe output but it is important to be aware of the dummy sequences so you do not mistake them as valid dark biome hits. This is also the point where any contigs not identified by BLAST are put into the darkbiome section of the pipeline.

```
blastn -db ntblastDB
-query fa_withdummy
-max_target_seqs 1
-max_hsps 1
-outfmt ”6 qseqid sseqid pident evalue staxids sscinames scomnames sskingdoms stitle”
-out name_unmapped.tsv
-num_threads 30
```

Joint Genome Institute (JGI) Query

At this point we can query the jgi server using the taxids identified by BLAST. Contigs matching Human, Mouse, Vertebrate, Viridiplantae, Artificial (synth or vector or Vector or artificial in the name) are removed and placed into the NF_OUT/unmapped/final/filter/contigs/blast/. In this folder you will also find the initial contigs that were made using spades. This step corresponds to the jgiJSON.ipynb and nb_jgiJSON.py scripts in the bin.

concat fasta for Bowtie2Index

At this step we will take all of the contigs that spade assembled and build bowtie2 indexes out of them so we can map reads to them. It is important that we are taking all of the contigs used in spades so reads that want to map back to ”junk” contigs have the opportunity. Additionally, since we get counts for all of the contigs we can use these in subsequent normal and dark biome calculations.

```
bowtie2-build –threads 20 condition.fasta condition
```

Map with bowtie2 and run mpileup

Now that we have indexes we can map using bowtie2. After mapping we sort and index these bams then run mpileup to get read counts for each basepair.

```
bowtie2 -q
-p 20
-x condition
-1 R1.fastq
-2 fin_R2.fastq
-S fout_condition.sam
samtools view -@ 20 -bS -o fout_conditiontmp fout_condition.sam
samtools sort -@ 20 fout_conditiontmp ¿ fout_condition.bam
samtools index fout_condition.bam
samtools mpileup -f fa fout_condition.bam ¿ fout_condition_pileup.txt
```

Dark Biome Dust Filter

At this point we have finished the normal biome. At the start of the Dark Biome we get all of the contigs that do not have a BLAST hit. Next, we run a dust filter from minimap2 that identifies repetitive regions, and use the created bed file to remove these contigs using seqkit.

```
sdust fa_withdummy > name_dust_mask.bed
```

2 Post Nextflow Scripts

After the nextflow pipeline run the bash script 2.0_MysteryMiner_Filter.sh to run all of the python scripts below

Filter darkGenome using LASTDB and BLASTX

Now that we have dark biome contigs free of overly repetitive sequences we can use the different levels/heirarchies of assembly (single sample, group/condition concatenated) to filter contigs found when all samples are assembled together. The logic here is that we want retain contigs that have a greater than %60 identitiy between assembled heirarchies. Briefly, this consist of making indexes out of the contigs in each heirarchy, then doing pairwise alignment between heirarchies. We use the single and group contigs to create a set of group contigs that are filtered by singles (Group_FS), then use the Group_FS contigs to filter the set of contigs made from all of the samples concatenated together. This gives you All_FG (All filtered by Group_FS). Finally, we protein BLAST (BLASTX) All_FG.
Contigs that have a BLASTX hit are identified with JGI similar and filtered similar to the normal biome, and can be located in the NF_OUT/unmapped/final/darkbiome/lastdb/FINAL_Tophits.
Contigs with no hits can be located in the NF_OUT/unmapped/final/darkbiome/lastdb/FINAL_Nohits folder.
This step was too complicated to execute in the nextflow pipeline and can be found in the 1.0-FilterDarkGenome.py and nb_1.0-FilterDarkGenome.py script in the bin.

Pileup coverage and normalization

In order to normalize samples by library size, we use the total mapped reads to the target genome from STAR to create a normalization factor for each sample. Normalization factors for each sample are calculated by finding the samples with the fewest mapped reads (norm_min), and dividing all of the other samples by the norm_min. We make the assumption that we can scale samples by the number of reads mapping to target genome and any unmapped reads that are not used in this calculation will have a negligible effect on normalization factors.

Next, using the mpileup output we calculate coverage for each contig by summing all of the counts for each base pair in a contig and dividing by the length of the contig.

Finally, we library normalize each contig coverage number in a sample by multiplying by the appropriate normalization factor of that sample.

This is done using the nb_2.0-PileupNormalize and nb_3.0-PileupDataFrame scripts in the bin.

Fastq and contig counting

Now we count all of the Fastq reads and contigs and create dataframes showing the amount. This is done in the nb_4.0-CountContigs scripts.
The amount of fastq reads removed at each step and for singles, groups, and all can be found in NF_OUT/unmapped/final/filter/fastq/fastq_amount_df.csv
The amount of regular biome contigs and amount removed at different filtering steps can be found in NF_OUT/unmapped/final/filter/contigs/blast/contigs_amount_df.txt'
The amount of dark biome contigs at each filtering step can be found in NF_OUT/unmapped/final/darkbiome/contigs_amount_dark_afterblast.txt

T-Test and quantification

In the script nb_5FilterTtest.py a generic T-Test is ran for every permutation of conditions and
ranks = ['superkingdom', 'kingdom', 'phylum', 'order', 'family', 'genus', 'species',"name"]
and
superkingdoms = ['Bacteria', 'Viruses', 'Eukaryota', 'NA', 'Archaea']
After running this script you can look in NF_OUT/FINAL_OUT for the final output. As an example, if you wanted to look for bacteria reads binned by species for condition A and B you would go to the final output and open the folder A_B_Bacteria_Bin_species and look in A_B_Bacteria_Bin_species_all_pileupCoverageNormalizedMatched.txt for the regular biome or A_B_Bacteria_Bin_species_dark_all_pileupCoverageNormalizedMatched.txt for the dark biome. There is also a folder for dark biome contigs in each comparison. The folder A_B_OUT_ALL_CONTIGS contains the individual contigs, fastas, and T-Test for that comparison.
Inside you will see output for doubledark
To create a custom query and T-Test open up the notebook 6.0-CustomTtest.ipynb once the other scripts are finished. This will allow you to create custom queries and remove taxonomies at different levels or select a particular taxonomy to look at. Detailed instructions are in the jupyter notebook.
A two-sided T-Test is ran using stats.ttest_ind from scipy and a padj value is calculated using the number of contigs or species in the dataframe using statsmodels.sandbox.stats.multicomp.fdrcorrection0 from statsmodels.
The significance threshold is set to 0.05 but can be changed inside the script.

Synthetic minibiome

We first used Polyester set to 10 replicates with two groups, error rate set to 0.005, paired end = TRUE, readlen = 100, and seed = 12.
We then generated the chr22 fasta using the default command from polyester
fasta_file = system.file('extdata', 'chr22.fa', package='polyester')
Next, we took the first 10 sequences from that and the first 10kb of the sequence. We then generated reads with no fold change difference and coverage set to 1000.
We next took the fastas of the sequences described in the paper and took the first 10kb from them. Next, we generated reads at multiple coverage levels from 1000 down to 0.01 in multiples of ten. We then combined these files.
We then used reformat.sh from bbmap to generate read quality scores from the output of polyester.
Finally, we combined the fastqs from human and minibiome using the "cat" command in bash and ran it through Mystery Miner.

Creating RDRP phylogeny tree

We first used BLASTX from the BLAST website on default settings on the contig to find similar hits. We then took the hits along with the RDRP contig performed multiple sequence alignment using CLUSTALW2 on default settings. https://www.ebi.ac.uk/Tools/msa/clustalo/. Next, we used the alignments to build the tree using Simple Phylogeny on default settings https://www.ebi.ac.uk/Tools/phylogeny/simple_phylogeny/.

RDRP contig amino acid sequence used in BLASTX
DEGSLESRGEDTKTRRSDQNPQGDIITDAEYAEVIHALKGYVWPDRSSNAELTSLLYQTGLGNTCTPECD
IFVKTKFSTIVSTCTSLYPKSNCHEDKASDVAFIHDVIRSELYTHTSVWDASPGYPFQIVYPTLLDLVDS
EPSALITLTLLLILRWGLTPHSQVRLMTAAELFEAKLTFLVRLFVKQEPHPVQKALDGRWRLVSSVPSHV
NVAARVLLGPQHRLNIRSCDYISPSIGLGLSDPMIQTHIRKAAMVEDSFGLVSSDQSGFDWRFYLLWADV
IAQVWVNLTCATGFWENAIRNYCYTMVFSYYVLSDGRIFGLLIPAARKSGDLDTGSGNSLHRIALNITIR
LWLKLERPSLVNRSVLPAMTMGDDCCESFGTRVDGPTLVEMFRQLGFKLTDVVIGSRNRFEFCSTRFEYD
GSWTITPLSWPRMLFRLLSQEPKQEFLDQFKYELRNLTGVYGGVNLRMLCLFLDRVGWKVPFDSPTNL