Radogest: random genome sampler for trees

supplemental information

Jacob S. Porter
jsporter@virginia.edu
Biocomplexity Institute and Initiative
University of Virginia

March 2020

1 Introduction

Radogest randomly samples fixed length nucleotide substrings (k-mers) for given taxonomic ids from a data store of genomes downloaded from the National Center for Biotechnology Information (NCBI). There is support for whole genomes, coding domain nucleotide data, and amino acid data. (However, each data type needs to be stored in its own directory because of limitations to Radogest's data structures.) Radogest is useful for generating kmers to train and analyze metagenomic classifiers, and it labels each kmer sampled with the taxonomic id that it represents. Radogest can generate data for any NCBI taxonomic id that is present in the NCBI data store that is of a conventional rank. Conventional ranks include superkingdom, kingdom, phylum, class, order, genus, and species.

2 Radogest Commands

2.1 Workflow

The following commands must be executed in order to generate k-mer samples. Once steps 1-5 have been completed, any number of data sets can be generated with the sample command.

1. **download**: download fasta files from NCBI.
2. **faidx**: generate fasta index files for each fasta file. This requires samtools faidx.
3. **index**: produce a Radogest genomes index data structure that includes information on genomes and taxonomy.
4. **tree**: create the taxonomic tree used by Radogest when sampling.
5. **select**: modify the Radogest index to select certain genomes for sampling.
6. **sample**: draw kmer samples and make a fasta file and a file of taxonomic ids. Requires bedtools random, a Radogest index after selection has been run, and a Radogest tree.

After step 4, tree, the following commands may be run to create mash distances. These are used for tree distance methods: TreeDist, LeafSelect (with clustering), and GenomeHoldoutTreeDistance.

4a. **sketch**: create genome mash sketches.
4b. **dist**: compute a distance matrix of genomes for each species.

2.2 Utilities

Radogest has three utility functions that are useful for manipulating fasta and taxonomic id files.

util_permute: Randomly permute and optionally split fasta records and their associated taxonomic ids.
util_chop: Cut up whole genomes with a sliding window and write out a fasta file and a taxonomic id file.
util_rc: Randomly take the reverse complement of DNA sequences in a fasta file.
util_subtree: Gets a list of all taxonomic ids underneath and including the given taxonomic id.
util_extract: Extract fasta records from a Radogest fasta file and taxid file that have the provided labels.
util_taxid: Give the species taxid, genome ids, and sample ids for a Radogest generated fasta file.

3 Genome Selection Algorithms

There are several genome selection strategies. These algorithms modify the Radogest index data structure so that only certain genomes underneath a taxonomic id will be used to generate k-mers during sampling. Genome selection is accomplished by using a post-order depth-first traversal of the taxonomic tree. This is given in Algorithm 1, SELECT.

SELECT requires a selection strategy S, and these are detailed in the following sections. If $\mathcal{T}$ is the set of nodes (taxonomic ids) in the taxonomic tree, and $\mathcal{G}$ is the set of genomes, then algorithm 1 has time complexity $O(|\mathcal{T}||\mathcal{G}|log(|\mathcal{G}|))$ in the worst case when choosing genomes by sorting. This is because the algorithm must visit every node on the taxonomic tree and choose genome labels for every genome at that node. The selection algorithms may sort the genomes, and, as is shown later, sorting has time complexity

Algorithm 1 Select: A depth-first post-order traversal.

```
r ← the root node of the taxonomic tree.
S ← the selection strategy algorithm to use.
I ← an index of genomes.                          ▷ A Python dictionary.
procedure SELECT(r, S, I)
    c_n ← the children of r
    for c_j ∈ c_n do SELECT(c_j, S, I)
    S(r, c_n, I)
```

$O(|\mathcal{G}|log(|\mathcal{G}|))$.

This procedure uses a genome index, which is a Python dictionary. The index was computed in the **index** step in the workflow. The index assigns each taxonomic id with a list of genomes and genome selection values. Suppose the index is $\mathcal{I}$, then for the taxonomic id t, $\mathcal{I}(t)$ gives a mapping of genome ids $\{g_i\}$ to selection values $\{v_i\}$, $\{g_1 : v_1, g_2 : v_2, \cdots, g_s : v_s\}$. The selection values $\{v_i\}$ are either boolean values or integers.

The following strategies randomly partition kmers from the same genome into training, validation, and test sets: AllGenomes (AG), ProportionalRandom (PR), TreeSelect (TS), LeafSelect (LS), and TreeDist (TD). GenomeHoldout strategies, on the other hand, separate whole genomes into training and testing data sets. These strategies begin with "GenomeHoldout". GenomeHoldout strategies can choose genomes either randomly or in sorted order in accordance with Algorithm 2. These strategies are unable to generate a validation (or third) data set. If a species has only one genome, then the kmers for that one genome are partitioned into training and test data sets.

The selection strategy algorithms first filter genomes in the following way. If a genome is duplicated between refseq and genbank, then the refseq genome is used. If a genome file does not have contigs, then it is excluded. This filtering step is done every time a genome set is generated, but this step is not shown in the pseudocode.

Several selection algorithms sort a set of genomes $\mathcal{G}$ by presumptive quality. Algorithm 2, SORT, gives the sorting algorithm that they use. When the index data structure is built, it stores whether the genome is a reference genome, a representative genome, or another genome. It stores the assembly level of the genome. This information is downloaded from the NCBI server. It computes the number of bases b_g and the number of contigs c_g in the genome by inspecting the genome file. The SORT algorithm uses this information.

The SORT algorithm first divides genomes into three sets of genome type: reference genomes, representative genomes, and other genomes. Each of these sets are divided into lists of genomes based on assembly level. Some genomes are more complete, and some are less complete. First by genome type and then from more complete to less complete genome assembly, the genomes are sorted in descending order by $\frac{b_g}{c_g}$. This ratio gives the number of bases divided by the number of contigs. The idea behind this is that if there are more contigs relative to bases, (many small contigs), then this genome could be lower quality. Since genomes are sorted first by reference status and by assembly level, higher quality, more complete genomes are chosen first. The algorithm returns the list of sorted genomes.

This algorithm runs in time $O(|\mathcal{G}|log(|\mathcal{G}|))$ because it sorts the genomes in descending order based on the numerical key $\frac{b_g}{c_g}$.

Algorithm 2 Sort: Sort the genomes.

$\mathcal{G}$ a set of genomes.
$\mathcal{I} \leftarrow$ a genome index where all genomes are initialized to False.
procedure SORT($\mathcal{G}$, $\mathcal{I}$)
    Divide the genomes $\mathcal{G}$ into three sets: reference genomes $\mathcal{R}_f$, representative genomes $\mathcal{R}_p$, and all other genomes $\mathcal{O}$.
    Let b_g be the count of bases for a genome g.
    Let c_g be the count of contigs for a genome g.
    $l \leftarrow$ an empty list
    for $\mathcal{S} \in \{\mathcal{R}_f, \mathcal{R}_p, \mathcal{O}\}$ in order **do**
        Divide $\mathcal{S}$ into sets of genomes by assembly level and order them from most complete to least complete.
        for each assembly level set in order of completeness **do**
            Sort the genomes in descending order by $\frac{b_g}{c_g}$
            Append this sorted list to l
    return The list l

The CLUSTER algorithm, shown in algorithm 3, can be used by strategies TreeDist and LeafSelect instead of sorting (algorithm 2) or random choosing to select genomes. It performs k-agglomerative clustering based on pre-computed mash distances. Then, it selects a representative genome for each cluster and returns the genomes sorted in descending order of the cluster size that they represent.

Algorithm 3 Cluster: Cluster the genomes. Radogest uses agglomerative clustering with an average linkage based on mash distances.

$\mathcal{G}$ a set of genomes.
$k \leftarrow$ the number of genomes desired
procedure CLUSTER($\mathcal{G}$, k)
    Perform a k-clustering of the genomes in $\mathcal{G}$
    $l \leftarrow$ an empty list
    for cluster $c \in \mathcal{C}$ **do** ▷ Choose a genome to represent the cluster.
        Find the genome in c with the least average distance to the other genomes in c
        Put that genome in l
    Sort l in order of decreasing cluster size
    return The list l

3.1 AllGenomes (AG)

Uses all non-redundant genomes. This set could be very large for high level taxonomic ids.

Algorithm 4 AG: Select all genomes.

1: $p \leftarrow$ parent taxonomic id.
2: $c_n \leftarrow$ a list of children of p.
3: $\mathcal{I} \leftarrow$ a genome index where all genomes are initialized to False.
4: **procedure** AG(p, c_n, $\mathcal{I}$)
5: Set all genomes in $\mathcal{I}(p)$ to True.

3.2 ProportionalRandom (PR)

At each level, this selection strategy chooses a fixed number of random genomes. For a parent node, only genomes selected by children nodes can be assigned to the parent.

Algorithm 5 PR: Choose a random set of genomes at each node.

$p \leftarrow$ parent taxonomic id.
$c_n \leftarrow$ a list of children of p.
$\mathcal{I} \leftarrow$ a genome index where all genomes are initialized to False.
$k \leftarrow$ the number of genomes to choose.
procedure PR(p, c_n, $\mathcal{I}$, k)
    if $|c_n| \neq 0$ **then** ▷ Inner node
        $l \leftarrow$ all genomes set to True in $\mathcal{I}(c_j)$ $\forall c_j \in c_n$
        Randomly choose k genomes in l.
        Set those genomes' values in $\mathcal{I}(p)$ to True.
    else ▷ Leaf node
        Randomly assign k genomes in $\mathcal{I}(p)$ to True.

3.3 LeafSelect (LS)

At the species level, the genomes are down selected, and the root of every subtree is given the genomes in all of the species (leaves) in the subtree. There are three ways that the genomes can be chosen: in sorted order, in random order, or by clustered order based on genome (mash) distances.

Algorithm 6 LS: At each species (leaf) node, choose at most k genomes. Propagate all chosen genomes up the tree.

$p \leftarrow$ parent taxonomic id.
$c_n \leftarrow$ a list of children of p.
$\mathcal{I} \leftarrow$ a genome index where all genomes are initialized to False.
$k \leftarrow$ the number of genomes to choose.
CHOOSE, the algorithm for choosing genomes.
procedure LS(p, c_n, $\mathcal{I}$, k, CHOOSE)
    if $|c_n| = 0$ **then** ▷ Leaf node
        Set genomes from CHOOSE($\mathcal{I}(p)$, k) to True in $\mathcal{I}(p)$
    else ▷ Inner node
        for $c_j \in c_n$ **do**
            for genome $g \in \mathcal{I}(c_j)$ that is set to True **do**
                Set g in $\mathcal{I}(p)$ to True

3.4 TreeSelect (TS)

At each node in the taxonomic tree, up to a fixed number k of genomes are chosen. Genomes are selected for a parent so that genomes from children are equally represented. Genomes can be either chosen in sorted order or chosen randomly.

Algorithm 7 TS: At each node, sort the genomes and choose at most k genomes.

$p \leftarrow$ parent taxonomic id.
$c_n \leftarrow$ a list of children of p.
$\mathcal{I} \leftarrow$ a genome index where all genomes are initialized to False.
$k \leftarrow$ the number of genomes to choose.
CHOOSE, the algorithm for choosing genomes.
procedure TS(p, c_n, $\mathcal{I}$, k, CHOOSE)
    if $|c_n| = 0$ **then** ▷ Leaf node
        Set genomes from CHOOSE($\mathcal{I}(p)$, k) to True in $\mathcal{I}(p)$
    else ▷ Inner node
        $l \leftarrow$ an empty list
        for $c_j \in c_n$ **do**
            Get the list l_j of genomes from CHOOSE($\mathcal{I}(c_j)$, k)
            Append l_j to l
        Set $i \leftarrow 0$ ▷ A counter for the number of genomes.
        Set $j \leftarrow 0$ ▷ A counter for choosing the genome list.
        while $i < k$ **do** ▷ Child genomes are represented equally.
            Pop off a genome in $l[j]$ and set it to True in $\mathcal{I}(p)$
            $i \leftarrow i + 1$
            $j \leftarrow j + 1 \mod |l|$

3.5 TreeDistance(TD)

At the species level, genomes are selected by clustering them based on mash distances. In inner nodes, genomes are chosen so that each subtree is equally represented. This is accomplished with the MERGE algorithm.

The MERGE algorithm is similar to the famous merge operation in the merge sort algorithm, but it can operate on more than two lists. It iterates through a list of lists and chooses elements from those lists in the order that they are in. It returns a flattened list.

This algorithm requires that the tree traversal method (depth-first search) passes into this algorithm a list of selected genomes from each child node. Thus, this algorithm returns a list of genomes.

Algorithm 8 TD: At species nodes, choose genomes based on clustering, and at inner nodes, choose genomes so that each subtree is equally represented.

$p \leftarrow$ parent taxonomic id.
$c_n \leftarrow$ a list of children of p.
$\mathcal{I} \leftarrow$ a genome index where all genomes are initialized to False.
$k \leftarrow$ the number of genomes to choose.
$g_l \leftarrow$ a list of lists of genomes
procedure TD(p, c_n, $\mathcal{I}$, k, g_l)
    if $|c_n| = 0$ **then** ▷ Leaf node
        Set genomes from CLUSTER($\mathcal{I}(p)$, k) to True in $\mathcal{I}(p)$
        return the list of genomes set to true
    else ▷ Inner node
        Shuffle g_l
        $g_l \leftarrow$ MERGE(g_l)
        Choose the first k genomes in g_l and set them to True in $\mathcal{I}(p)$
        return g_l

Algorithm 9 Merge: Merge two lists so that elements are chosen in the order that they appear in the sublists.

```
g_l ← A list of lists.
procedure MERGE(g_l)
    g_n ← an empty list
    while there are unchosen elements in g_l do
        for l ∈ g_l do
            Append the first unchosen element from l to g_n
    return g_n
```

3.6 GenomeHoldoutSpeciesLeaf (GHSL)

Each species is alternatively labeled as either part of the training or the testing data. A maximum number of genomes for each species is chosen either sorted by quality or randomly. The training or testing label for each genome is propagated up the taxonomic tree. This strategy does not generate a validation data set.

Algorithm 10 GHSL: Holdout species for training and testing. Propagate all choices up the tree.

```
p ← parent taxonomic id.
c_n ← a list of children of p.
I ← a genome index where all genomes are initialized to False.
k_1 ← the number of genomes to choose for training.
k_2 ← the number of genomes to choose for testing.
s ← the training or testing state. s ∈ {0, 1}.
r ← an indication for sorting or randomizing genomes.
procedure GHSL(p, c_n, I, k_1, k_2, s, r)
    if |c_n| = 0 then                                   ▷ Leaf node
        Put the genomes in I(c_n) in order r
        Set up to k_s genomes in I(c_n) to state s
        Alternate state s
    else                                                ▷ Inner node
        for c_j ∈ c_n do
            for genome g ∈ I(c_j) do
                Set g in I(p) to the same label as in I(c_j)
```

3.7 GenomeHoldoutSpeciesTree (GHST)

Each species is alternatively labeled as either part of the training or the testing data. A maximum number of genomes for each species is chosen either sorted by quality or randomly. At each level of the tree, each genome is labeled as either part of the train or the test set up to a maximum number of genomes. Genomes from each child are represented equally. This strategy does not generate a validation data set.

Algorithm 11 GHST: Select species for training and testing. At each inner node, select genomes for training or testing so that each child is represented equally.

$p \leftarrow$ parent taxonomic id.
$c_n \leftarrow$ a list of children of p.
$\mathcal{I} \leftarrow$ a genome index where all genomes are initialized to False.
$k_1 \leftarrow$ the number of genomes to choose for training.
$k_2 \leftarrow$ the number of genomes to choose for testing.
$s \leftarrow$ the training or testing state. $s \in \{0, 1\}$.
$r \leftarrow$ an indication for sorting or randomizing genomes.
procedure GHST(p, c_n, $\mathcal{I}$, k_1, k_2, s, r)
    if $|c_n| = 0$ **then** ▷ Leaf node
        Put the genomes in $\mathcal{I}(c_n)$ in order r
        Set up to k_s genomes in $\mathcal{I}(c_n)$ to state s
        Alternate state s
    else ▷ Inner node
        for each state s_i **do**
            Select an equal number of genomes from each child with state s_i and given them that state in $\mathcal{I}(p)$
            Randomly assign singleton state genomes to another state in $\mathcal{I}(p)$ if there are too many singleton genomes.

3.8 GenomeHoldoutGenomeLeaf (GHGL)

At each species leaf, full genomes are labeled as either part of the training or the testing set. This labeling is propagated up the tree. If there is only a single genome for a species, then k-mers from the genome are partitioned into either the training or the testing sets. This strategy does not generate a validation data set.

Algorithm 12 GHGL: Holdout whole genomes for training or testing at every species node. Propagate all choices up the tree.

$p \leftarrow$ parent taxonomic id.
$c_n \leftarrow$ a list of children of p.
$\mathcal{I} \leftarrow$ a genome index where all genomes are initialized to False.
$k_1 \leftarrow$ the number of genomes to choose for training.
$k_2 \leftarrow$ the number of genomes to choose for testing.
$s \leftarrow$ the training or testing state. $s \in \{0, 1\}$.
$r \leftarrow$ an indication for sorting or randomizing genomes.
procedure GHGL(p, c_n, $\mathcal{I}$, k_1, k_2, s, r)
    if $|c_n| = 0$ **then** ▷ Leaf node
        if $|\mathcal{I}(c_n)| = 1$ **then** set the $g \in \mathcal{I}(c_n)$ to the singleton state.
        else
            Let $|\mathcal{G}_s|$ be the number of genomes selected for state s
            for $g \in \mathcal{I}(c_n)$ in r order **do**
                if $|\mathcal{G}_s| < k_s$ **then**
                    Set g to s
                    Alternate state s
    else ▷ Inner node
        for $c_j \in c_n$ **do**
            for genome $g \in \mathcal{I}(c_j)$ **do**
                Set g in $\mathcal{I}(p)$ to the same label as in $\mathcal{I}(c_j)$

3.9 GenomeHoldoutGenomeTree (GHGT)

At each taxonomic node, full genomes are labeled as either part of the training or the testing set. If there is only a single genome for a species, then k-mers from the genome are partitioned into either the training or the testing sets. Genomes from each child are represented equally. This strategy does not generate a validation data set. No more than some limit of genomes are chosen for training or testing.

Algorithm 13 GHGT: Select whole genomes for training or testing at each taxonomic node.

$p \leftarrow$ parent taxonomic id.
$c_n \leftarrow$ a list of children of p.
$\mathcal{I} \leftarrow$ a genome index where all genomes are initialized to False.
$k_1 \leftarrow$ the number of genomes to choose for training.
$k_2 \leftarrow$ the number of genomes to choose for testing.
$s \leftarrow$ the training or testing state. $s \in \{0, 1\}$.
$r \leftarrow$ an indication for sorting or randomizing genomes.
procedure GHGT(p, c_n, $\mathcal{I}$, k_1, k_2, s, r)
    if $|c_n| = 0$ **then** ▷ Leaf node
        if $|\mathcal{I}(c_n)| = 1$ **then** set the $g \in \mathcal{I}(c_n)$ to the singleton state.
        else
            Let $|\mathcal{G}_s|$ be the number of genomes selected for state s
            for $g \in \mathcal{I}(c_n)$ in r order **do**
                if $|\mathcal{G}_s| < k_s$ **then**
                    Set g to s
                    Alternate state s
    else ▷ Inner node
        for each state s_i **do**
            Select an equal number of genomes from each child with state s_i and given them that state in $\mathcal{I}(p)$
        Randomly assign singleton state genomes to another state in $\mathcal{I}(p)$ if there are too many singleton genomes.

3.10 GenomeHoldoutTreeDistance(GHTD)

This selection strategy is similar to the TreeDistance strategy, but it maintains lists of test set and training set genomes as well as singleton genomes. Each genome is marked with a state representing whether it is in the training set, the test set, or both (singleton genomes). A different number of genomes can be chosen for the training set and the testing set.

Algorithm 14 GHTD: At species nodes, choose genomes based on clustering, and at inner nodes, choose genomes so that each subtree is equally represented.

$p \leftarrow$ parent taxonomic id.
$c_n \leftarrow$ a list of children of p.
$\mathcal{I} \leftarrow$ a genome index where all genomes are initialized to False.
$k \leftarrow$ the number of genomes to choose
$g_l \leftarrow$ a list of lists of genomes
procedure GHTD(p, c_n, $\mathcal{I}$, k, g_l)
    if $|c_n| = 0$ **then** ▷ Leaf node
        If there is a single genome, set its state to 2 (singleton).
        Otherwise, alternatively set genomes from CLUSTER($\mathcal{I}(p)$, k) to $s \in \{0, 1\}$ in $\mathcal{I}(p)$
        return a list of genome lists for each state
    else ▷ Inner node
        g_n an empty list
        for $s \in \{0, 1, 2\}$ **do**
            $g_s \leftarrow$ list for s in g_l
            Shuffle g_s
            $g_s \leftarrow$ MERGE(g_s)
            Choose the first k genomes in g_l and set them to s in $\mathcal{I}(p)$
            Append g_s to g_n
        return g_n

4 Sampling Algorithm

Once genome selection is complete and the Radogest index is updated, then Radogest can generate k-mers in fasta files. It uses algorithm 15 SAMPLE. This algorithm can be run in parallel on multiple taxonomic ids, or it can be run in parallel on a single taxonomic id where each genome is sampled from in parallel.

If the selection strategy is not a GenomeHoldout strategy, then the returned list l is randomly partitioned into training, validation, test sets in proportion to user specification. For these selection strategies, k-mers can be generated randomly, or the whole genome can be cut up with a sliding window. Using a random approach, the number of k-mers drawn from a genome g is directly proportional to the total number of bases in the genome. So, if m is the total length of all genomes to be drawn from and n is the total number of k-mers requested for taxonomic id t, then there are approximately $\frac{|g|}{m}n$ k-mers sampled from genome g. A random draw of a k-mer consists of generating a starting position in the genome string uniformly at random with replacement. If a k-mer cannot be generated because the starting position would generate a string that is too short, then that starting position is not used. It is possible to have duplicate k-mers because the starting position can be randomly generated multiple times.

A sliding window approach can be used instead of random starting positions. If this is done, then n, the number of k-mers requested, is ignored. A stride parameter can be used to determine the stride of the sliding window. This approach gives a variable number of k-mers.

If the selection strategy is a GenomeHoldout strategy, then SAMPLE is called separately on training set genomes and on testing set genomes. The k-mers are generated using a sliding window with a parameterized stride. If the genome is a singleton genome, then k-mers are generated with a sliding window and k-mers are randomly assigned to either the training or the testing data set. For each taxonomic level, the k-mers given to the training or the testing set will always be the same for singleton genomes because the random number generator that assigns them has a fixed seed.

Algorithm 15 Sample: Sample k-mers from genomes for a given taxonomic id t.

$t \leftarrow$ a taxonomic rank
$n \leftarrow$ the total number of k-mers to sample
$k \leftarrow$ the length of the k-mer
$s \leftarrow$ a variable that determines if genomes should be cut up with a sliding window or randomly sampled
$h \leftarrow$ an optional genome size threshold for controlling random draws
$\mathcal{I} \leftarrow$ a Radogest index with selected genomes
procedure SAMPLE(t, n, k, s, h, $\mathcal{I}$)
    $l \leftarrow$ an empty list of k-mers (and ids)
    $m \leftarrow$ the total length of all genomes to sample from
    for children c_j of t **do**
        for genome g selected in $\mathcal{I}(c_j)$ **do**
            if s indicates random and $|g| > h$ **then**
                Add approximately $\frac{|g|}{m}n$ random k-mers from g to l
            else
                Use a sliding window to add k-mers from g to l.
    return l