Reducing noise in protein multialignments

Marina Herrera Sarrias

January 6, 2019

Abstract

In this research a noise-reduction method is implemented in protein multialigments to evaluate its impact on phylogenic inference.The data used in this project is a reduced data set from the original data used by the creators of TrimAI. To test the data the program fastprot was used to obtain the distance matrices and fnj to infer a phylogenic tree for each alignment. Dendropy has been used to measure the symmetric distance between the inferred tree and the reference tree.

Contents

1 Introduction

The implemented noise-reduction method evaluates a multialigment column as noisy if there are more than 50% indels, at least 50% of the amino acids are unique and non of the amino acids appear more than twice. Each column takes the form of a sequence of amino acids given a position of the aligned sequences.

The reduced data set used in this project is composed by six different directories, each directory contains a reference tree and 300 alignments created by evolving sequences along the reference tree. Each pair of directories present an average amount of mutations of 0.5, 1.0 and 2.0 per sequence site. Also, per each mutation rate there is one symmetric reference tree and one asymmetric reference tree.

For validating the effect of reducing reducing

2 Results and Discussion

3 Methods and Materials

4 run_program.py

```python
import sys
import glob
import os
import json
import subprocess
from collections import Counter

from Bio.Seq import Seq
from Bio import SeqIO, AlignIO, Align, SeqRecord
import dendropy
from dendropy.calculate import treecompare
from Bio import Phylo

"""
This script will perform all the program's required tasks.
It will first create a list of all the data directories where
   we will reduce noise,
new directories for storing the reduced alignments and an empty
    dictionary
with key 'main directory name' (e.g asymmetric_0.5) and which
   will
later contain a nested dictionary:
{
    'filename': [
    original_alignment_symmetric_distance,
    noise_reduced_alignment_symmetric_distance,
    noise_reduction_ratio_
    ]
    ...
}
 with 'filename' e.g: s001.align.1.msl
and original_alignment_symmetric_distance,
   noise_reduced_alignment_symmetric_distance,
   noise_reduction_ratio_:
    distance of the reference tree with the original alignment
       tree and
    distance of the reference tree with the noise reduced
       alignment tree
    ratio between the difference of number of columns between
       the original and reduced alignment and the number of
       rows
    of the original alignment.
```

```python

The program will start parsing through the raw data, it will first read the reference tree,
set the keys of the dictionary and write the noise reduced alignment file using the
perform_noise_reduction function, afterwards it will generate inferred tree files, for both:
the raw data and the noise reduced data using the function computing_and_writing_alignment_tree.
Lastly it will compute the distance between the reference tree and both inferred tree cases
(with and without noise reduction). The file names (key), distances and ratios (key values) will be nested
in the dictionary.

The program will write a json file in the results folder containing the dictionary.
"""


def noise_filter(column):
    """
    noise_filter is a function that determines whether an alignment column is noisy or not.
    :param column: multiple sequence alignment column.
    :return: boolean.
    """
    indel = '-'
    amino_counter = Counter(column)
    if column.count(indel) > len(column) / 2:
        return True
    if len([key for key, value in amino_counter.items() if value == 1]) >= len(column) / 2:
        return True
    if max(amino_counter.values()) <= 2:
        return True
    return False


def reduce_noise(multiple_seq_alignment, _alignment_path):
    """
    reduce_noise is a function that filters all noisy columns.
    :param multiple_seq_alignment: MultipleSeqAlignment.
    :param _alignment_path: msa path.
    :return: a MultipleSeqAlignment.
    """
    noise_free_column_list = []
    new_aligned_list = []
    alignment_filename_ = _alignment_path.split('/')[-1]

    for i in range(multiple_seq_alignment.get_alignment_length()):
        if not noise_filter(multiple_seq_alignment[:, i]):
            noise_free_column_list.append(multiple_seq_alignment[:, i])
    for i in range(len(multiple_seq_alignment)):
        new_aligned_list.append(''.join([x[i] for x in noise_free_column_list]))
    record_list_ = Align.MultipleSeqAlignment([
        SeqRecord.SeqRecord(Seq(new_seq), id=record.id, name=record.name, description=record.description)
        for new_seq, record in zip(new_aligned_list, multiple_seq_alignment)
     ])
    noise_reduction_ratio_ = (multiple_seq_alignment.get_alignment_length()
                              - record_list_.get_alignment_length()) / multiple_seq_alignment.get_alignment_length()

    if multiple_seq_alignment.get_alignment_length() == record_list_.get_alignment_length():
        print(f'{alignment_filename_}'+ '>> WARNING: no noise reduction')
    if 0 < record_list_.get_alignment_length()  < multiple_seq_alignment.get_alignment_length() / 2:
        print(f'{alignment_filename_}'+ '>> WARNING: more than 0.5 of the sequence is noise')
    return record_list_, noise_reduction_ratio_



def perform_noise_reduction(alignment_path_):
    """
    perform_noise_reduction is a function that takes the parameter alignment path and applies
    the reduce_noise function.
    :param alignment_path_: msa path.
    :return: a MultipleSeqAlignment.
    """
    with open(alignment_path_, mode='r') as aligned_file:
        my_sequence_recorded = AlignIO.read(aligned_file, 'fasta')
    return reduce_noise(my_sequence_recorded, alignment_path_)


def computing_and_writing_alignment_tree(msa_filename, tree_outfile_):
    """
    computing_and_writing_alignment_tree is a function that pipes two processes,
    it first creates a distance matrix for each msa using fastprot,
    an then infers an alignment tree using fnj. The function writes the inferred
    tree in a file per msa.
    :param msa_filename: msa path.
    :param tree_outfile_: inferred tree path.
    :return: file containing the inferred tree.
    """
    args= "cat " + msa_filename + " | fastprot | fnj -O newick -o " + tree_outfile_
    child_noise_reduced = subprocess.Popen(args, shell=True)
    child_noise_reduced.wait()


def compare_trees(ref_tree, inferred_tree, is_bipartitions_updated=False):
    """
    compare_trees is a function that computes the simmetric difference between
    the reference tree and the inferred tree.
    :param ref_tree: reference tree read using dendropy.
    :param inferred_tree: inferred tree read using dendropy.
    :param is_bipartitions_updated: recalculates bipartitions.
    :return: (int) symmetric distance between the reference and inferred tree.
    """
    sd = treecompare.symmetric_difference(ref_tree, inferred_tree)
    return sd


def compute_distance_between_trees(inferred_tree_file_path, reference_tree_):
    """
    compute_distance_between_trees is a function that reads inferred trees using
    dendropy and computes the distance using the function compare_trees
    :param inferred_tree_file_path: path
    :param reference_tree_: reference tree read using dendropy.
    :return: (int) symmetric distance between the reference and inferred tree.
    """
    with open(inferred_tree_file_path, mode='r') as reduced_noise_tree_file:
            reduced_noise_tree_str = ''.join(list(reduced_noise_tree_file))
            reduced_noise_tree = dendropy.Tree.get_from_string(
                reduced_noise_tree_str,
                schema="newick",
                taxon_namespace=tns
            )
    return compare_trees(reference_tree_, reduced_noise_tree)


if __name__ == '__main__':
    args_in = sys.argv[1:]
    if len(args_in) > 0:
        print('         ')
        sys.exit('WARNING: run_program do not require any input.')
    print('processing data...')

    original_dir = './data/raw_data'
    reduced_dir = './data/noise_reduced_data'
    result_dir = './results'
    directories = []
    tns = dendropy.TaxonNamespace()

    for folder in glob.glob(original_dir +'/*'):
        if len(glob.glob(original_dir + '/*')) == 0:
            sys.exit('empty data directory, please verify your data is on the right directory')

        sub_folder_name = folder.split('/')[-1]
        directories.append(sub_folder_name)

```

```python
    # this will contain all the symmetric distances between inferred trees
    # and reference trees, for all different data subdirectories
    compare_trees_dictionary = dict()

    for directory in directories:  # directories = [asymmetric_0.5, asymmetric_1.0, ...]
        # create asymmetric_0.5, asymmetric_1.0, ... subdirectories into the reduced_data directory
        # and getting the reference tree for each subdirectory of alignments.
        new_folder_path = os.path.join(reduced_dir, directory)
        os.makedirs(new_folder_path, exist_ok=True)
        original_dir_path = os.path.join(original_dir, directory)
        if len(glob.glob(original_dir_path + '/*')) == 0:
            sys.exit('empty data sub directory, please verify your data is on the right directory')

        reference_tree_path = glob.glob(original_dir_path +'/*.tree')[0]
        with open(reference_tree_path, mode='r') as ref_tree_file:
            ref_tree_str = ''.join(list(ref_tree_file))
            reference_tree = dendropy.Tree.get_from_string(
                ref_tree_str,
                schema="newick",
                taxon_namespace=tns
        )
        folder_dict_key_name = original_dir_path.split('/')[-1]

        compare_trees_dictionary[folder_dict_key_name] = dict()

        for alignment_path in glob.glob(original_dir_path + '/*.msl'):
            if os.stat(alignment_path).st_size == 0:
                sys.exit(f'{alignment_path}'+ '>> ERROR: empty msl file.')
            # performing noise reduction and writing the alignment
            # into the corresponding noise_reduction directory
            record_list, noise_reduction_ratio = perform_noise_reduction(alignment_path)
            if record_list.get_alignment_length() == 0:
                sys.exit(f'{alignment_path}'+ '>> WARNING: all columns are noisy.')
            reduced_alignment_name = alignment_path.split('/')[-1]
            reduced_filename_out = os.path.join(new_folder_path, reduced_alignment_name)
            AlignIO.write(record_list, reduced_filename_out, "fasta")

            # computing and writing noise reduced alignment trees
            tree_outfile_reduced = reduced_filename_out[:-3] + 'tree'
            computing_and_writing_alignment_tree(reduced_filename_out, tree_outfile_reduced)
            if os.stat(tree_outfile_reduced).st_size == 0:
                tree_file_name = tree_outfile_reduced.split('/')[-3:]
                sys.exit(f'{tree_outfile_reduced}' + '>> ERROR: empty tree file.')

            # computing and writing original alignment trees
            alignment_name = alignment_path.split('/')[-1]
            filename_out = os.path.join(original_dir_path, alignment_name)
            tree_outfile = filename_out[:-3] + 'tree'
            computing_and_writing_alignment_tree(alignment_path, tree_outfile)
            if os.stat(tree_outfile).st_size == 0:
                tree_file_name = tree_outfile.split('/')[-3:]
                sys.exit(f'{tree_outfile}' + '>> ERROR: empty tree file.')

            # computing distance between ref tree and inferred trees
            noise_reduced_distance = compute_distance_between_trees(tree_outfile_reduced, reference_tree)
            original_distance = compute_distance_between_trees(tree_outfile, reference_tree)

            alignment_key_name = alignment_name[:-3]
            compare_trees_dictionary[folder_dict_key_name][alignment_key_name] = (
                original_distance, noise_reduced_distance,
                  round(noise_reduction_ratio, 2)
            )

    distance_results_path = os.path.join(result_dir, '
      distance_result_dict')
    with open(distance_results_path, 'w') as result_dir_file:
        json.dump(compare_trees_dictionary, result_dir_file)

    print('your data has been processed!')
```