Pangenome_rf.py instructions

Alan Beavan

1. Gather materials
2. Process matrix
3. Perform random forest
4. Simplify the importance matrix [optional]
5. Convert to a cytoscape/gephi format [optional]
6. Direct the edges of the network so that inhibitors are indicated as different from promoters [optional]
7. Calculate the D statistic for all genes [optional]
8. Make an SQL database to easily find statistics about your results [optional]

##

1. Gather materials.
   a. Clone the github repository using

      ```
      git clone https://github.com/alanbeavan/pangenome_rf
      ```

   b. Infer the pangenome using panaroo (for details see their documentation). The file we need is gene_presence_absence.csv. The github repository also features a gene presence absence table that can be used to get to grips with the software.

   c. Install python 3.9 or above. This can be done by installing from source locally or using anaconda. To install using anaconda, create a python 3.10 environment.

      ```
      conda create --name py10 python=3.10
      ```

      Then activate it

      ```
      conda activate py10
      ```

   d. Install python packages in your new anaconda python 3.10 environment. If you installed a local version of python, you can use pip3 to do this instead. Here are the anaconda commands.

      ```
      conda install pandas
      conda install -c anaconda scikit-learn
      ```

      I don't think there are any more but if the program produces an error, read it. It might indicate that you are missing a module, which you can then install using anaconda (please also let me know which you were missing). You should now be ready to move on to matrix processing.

2. Process matrix
   a. The panaroo presence absence matrix is a bit of a mess so you'll need to run process_matrix.py to clean it up.

      ```
      python3 process_matrix.py <panaroo_presence_absence_matrix> <output_file>
      ```

("<" and ">" usually represent options that you can change in case you didn't know). My suggestion would be to name the output file collapsed_matrix.csv and I will refer to it as this from now on. The program will generate 6 files. These are listed below.

collapsed_matrix.csv – A version of the presence absence matrix that panaroo created where, instead of the gene name(s), 1 indicates presence and 0 indicates presence. Also, gene families with identical presence absence patterns have been collapsed (they will be called family_group_n where n is a number). Genomes with identical gene profiles have also been collapsed into genome_group_n. Single copy genes and genes found in all genomes are removed from this matrix as they are worse than useless.
constant_genes.txt – A list of genes found in all genomes in your dataset
core_genes.txt – A list of genes found in >= 95% of genomes in your dataset
singletons.txt – A list of genes found in only one genome of your dataset
non_unique_genes.txt – A tab separated table with the first column being the list of gene family groupings created by the program and the second column being a comma separated list of the genes in the group.
non_unique_genomes.txt – A tab separated table with the first column being the list of genome groupings created by the program and the second column being a comma separated list of the genomes in the group.

3. Perform random forest
   a. First, I would recommend familiarising yourself with the options by running

```
python3 pangenome_rf.py -h
```

   This will print the usage of the program.

   b. Next, I would usually run a short test to ensure that everything you've done up to now is working (10 minutes walltime on 1 core of 1 node would be fine). Use this command:

```
python3 pangenome_rf.py -n 1 -d 1 -m collapsed_matrix.csv -pres 1
-abs 1 -o test
```

   When the job is finished you should see output of at least a few lines that look something like this:

```
gene number     1       out of  11018
```

   The number that it is out of will depend on how many genes are in your dataset. It may also have written some output in the form of imp.csv and performance.csv in a directory "test". This will only have happened if it got to 1000 genes though, so don't worry if it has not.

   c. Run the random forest program. The usage is as follows:

```
usage: pangenome_rf.py [-h] [-n NTREES] [-d DEPTH] [-m FILENAME]
[-pres MIN_PRESENT] [-abs MIN_ABSENT] [-o OUTPUT] [-r] [-t
NTHREADS]

options:
  -h, --help            show this help message and exit
  -n NTREES, --n-trees NTREES
                        number of trees in the forest
  -d DEPTH, --depth DEPTH
                        max depth of trees in the forest
  -m FILENAME, --matrix FILENAME
                        matrix file
```

```
  -pres MIN_PRESENT, --min-present MIN_PRESENT
                        minimum percentage of genomes featuring a
gene for it to be analysed (5 = 5 percent, not 0.05)
  -abs MIN_ABSENT, --min-absent MIN_ABSENT
                        minimum percentage of genomes missing a
gene for it to be analysed (5 = 5 percent, not 0.05)
  -o OUTPUT, --output-directory OUTPUT
                        destiny directory for results
  -r, --randomise       randomise the gene presence and absence
(null hypothesis)
  -t NTHREADS, --n-threads NTHREADS
                        number of threads (default - 1)
-c CHECKPOINT, --checkpoint CHECKPOINT
                        continue from checkpoint? provide the
                        number of genes that have been completed
```

You need to decide 5 main parameters.

I. NTREES – the number of trees in the forest – It is difficult to know what the best number I have been surprised by how little the accuracy increases with more trees. You may want to try a series so that you can assess whether adding trees will really increase your accuracy, or is a waste of time. Perhaps trying 100, 500 and 1,000 would be a good place to start.

II. DEPTH – the number of nodes in the decision trees. By contrast I have been surprised by how much increasing the depth increases the accuracy. I am running a series of 2, 4, 8 and 16. I am yet to assess the best depth to use. Generally, the higher the depth the higher the accuracy on the training set, but the more likely you are to overfit the model, which can lead to poor precision or recall. This is worth reading about.

III. MIN_PRESENT – the minimum number of genomes that the gene must be present in to include in the analysis as a percentage. If it is too high, we are throwing away useful predictors and potentially interesting results. If it's too low, we are including uninformative genes in our analysis which will increase the number of trees/depth needed. I used 1% for this, which I think is sensible

IV. MIN_ABSENT – the minimum number of genomes that the gene must be absent in. The same logic as above led me to 1% as a good value.

V. NTHREADS – the number of parallel processes to be run simultaneously during the random forests. Hopefully it scales more or less linearly. Make sure that you ask for the correct number of cores if submitting in a job script.

The only other option to consider is whether randomise (-r). This should only be done as a null hypothesis to compare your actual results to. i.e. how much signal is there in a random dataset of the same size and with the same gene frequencies as your dataset?

When you have decided these parameters, run something along the lines of:

```
python3 pangenome_rf.py -n <1000> -d <8> -m <collapsed_matrix.csv>
-pres <1> -abs <1> -o <output>
```

It's worth keeping an eye on the output directory. Every 1000 genes it will update the imp.csv and performance.csv tables, so you can look into the files to see how far it has gotten.

Essentially, you're now done and simply have to parse your results as you see fit. Here are some things I have found useful.

4. Simplify importance matrix [optional]

For each gene, many other genes will have a non-0 importance, meaning they improve the accuracy of the prediction. However, some of these will be very low. They might be real, but they might be a statistical coincidence of our data. Even if they are real, they are probably less interesting than strong predictors and will make our visualisation difficult. Thus, a simplified matrix can be created by setting an arbitrary cut-off value, below which relationships should be ignored (in practice, converted to 0). You must decide a cut-off that is low enough that we are minimising the loss of results, but high enough that the network can be effectively visualised (and doesn't take up too much RAM). Run the following to achieve this.

```
python3 simplify_imp.py <threshold> <input_imp.csv> <output_file>
```

The threshold should be actively considered after looking at the distribution of importances or the network visualisation. I used 0.01.

5. Convert to a cytoscape/gephi format [optional]

   Gephi, a program for network visualisation can read the matrix in the form it's in when the program finishes. Cytoscape cannot. To convert to a format that both can read, and that some downstream programs I have written work on, run convert_to_cytoscape.py using the following.

```
python3  convert_to_cytoscape.py <input imp.csv>
<output_network.csv>
```

   The new file will be a list of lines, each representing an edge of the network.

6. Direct the edges of the network so that inhibitors are indicated as different from promoters [optional]

   Until now, the edges represent the importance in predicting the presence of a gene; not whether the gene is promoted (more likely to be present) by the predictor, or inhibited (less likely to be present). To remedy this, the program - direct_network.py – reads each interaction in the cytoscape format csv file and infers whether the predictor makes the gene more likely to be present (in which case the interaction remains a positive one) or less likely (in which case the interaction is changed to a negative one). You need to point the program to gene presence absence matrix as well as your cytoscape format csv and an output file.

```
python3 direct_network.py <input cytoscape csv>
<collapsed_network.csv> <output>
```

7. Calculate the D statistic for all genes

   Analyses of this type are generally interested in relationships of genes that transfer horizontally, rather than those that cooccur purely down to a shared ancestry. The D statistic can be used to estimate how closely the distribution of presence/absence of a gene matches the phylogeny. You will need a rooted phylogeny of your strains (you could use IQTree, Fasttree or a myriad of other programs) and a comma separated list of the genes you are interested in (e.g. all genes involved in an edge) with the suffix nodes_in.csv.

   You need to have R installed with the following libraries (You can use install.packages())

   caper
   phytools
   getopt
   future
   flock
   stringr
   flock

   To calculate D for all these genes, run

```
Rscript calculate_d.R -a <path> -t <treefile> -g
<presence_absence_matrix> -c <ncores>  -o <prefix>
```

prefix should be whatever comes before “nodes_in.csv” in your genes list. The output is a file -<prefix>nodes.tsv which is tab separated values table of each gene its D value.
A word of warning – the program cannot deal with “~” which comes up a lot in prokka annotations. To circumnavigate this problem, replace the “~” with some character or set of characters and then convert back after the program has completed. This program might take a couple of days to run.

8. Make an SQL database to easily find statistics about your results

To really scrutinise your results, you might find it useful to generate an sql database with some statistics associated (Figure 1). How you go about this is entirely down to you, but if ou like, you can follow the pipeline I created, with instructions here. First you need to fully describe your nodes (with performance metrics and D). To do this run

```
python3 describe_nodes.py -p <PERFORMANCE> -d <D_TABLE> -o
<OUTPUT_FILE>
```

Where performance is the PERFORMANCE table output from pangenome_rf.py and D_TABLE is the D table outputted from calculate_d.R
To describe your edges full run:

```
python3 describe_edges.py -m <MATRIX_FILE> -n <NETWORK_FILE> -o
<OUTPUT_FILE>
```

where MATRIX_FILE is your collapsed matrix and NETWORK_FILE is your simplified, directed network. Allow some time because this program has to measure the genome occupancy of each gene in an edge both in the presence and absence of the other gene in the edge.
Once these have finished, you can make the SQL database – you need to have SQLite3 installed, although it may work using other distributions of SQL (not tested).

```
python3 make_sql_database.py [-h] -e <EDGES_FILE> -n <NODES_FILE>
-o <OUTFILE>
```

where EDGES_FILE and NODES_FILE are the outputs of describe_edges.py and describe_nodes.py respectively. You now have an SQL database you can query. Some useful queries are included in the file list_of_queries.sql but you are welcome to design your own or modify these.

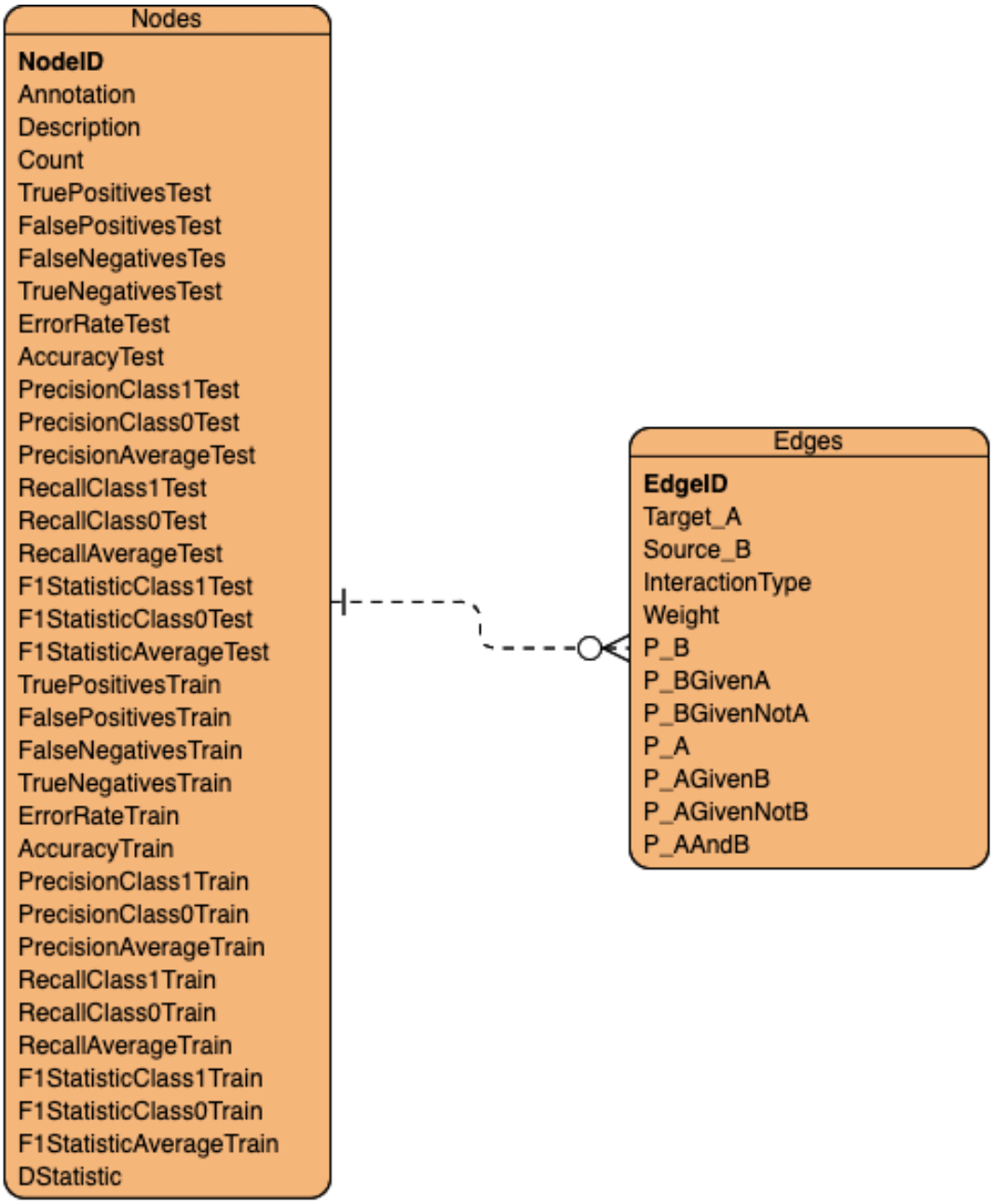

Figure 1: SQLite3 relationship diagram. The two tables are linked in a one to many relationship indicated by the connection. The names of tables and columns are indicated with the primary key in bold. P_ here means probability of.

Have fun