Bioremediation potential prediction pipeline

Libe Renteria

February 2024

1 Introduction

This pipeline outlined the steps for creating a Random Forest model that predicts the biormediation potential of bacteria present in specific environmental conditions. We will create a bioremediation gene database, a bacterial genome database (in our case, specific to the *Pseudomonas putida* database) and a data exploration and random forest model that will help us in the selection of environmental conditions that will maximize the probability of finding bioremediation related bacteria.

2 Custom databases

2.1 Gene database

We will start creating a gene database with genes related with bioremediation. For this, we will retrieve experimentally proven genes related to bioremediation; in our case, as we will study the *Pseudomonas putida* database) species, we find genes that have been found in bacteria from this species. The created database will contain the following columns:

- Gene group: We will group the genes depending on their general function. Some examples of this: Hydrocarbon degradation, aromatic compound degradation, heavy metal resistance genes...
- General function: Description of gene group. If the group is aromatic compound degradation, its general function would be: "This group focuses on genes involved in the degradation of aromatic hydrocarbons like benzene, toluene, xylene, and polycyclic aromatic hydrocarbons (PAHs). These are common pollutants found in industrial waste and oil spills."
- Gene name: Abreviation of bioremediation related gene
- Gene description: Full gene name and description of the specific gene function.
- Nt: nucleotide sequence of the gene
- Aa: protein sequence of the gene
- Papers: Link to paper where the bioremediation function is demonstrated
- Species: Species from which the gene sequences are retrieved
- Aa link: link to aa sequence in database.
- Nt link: link to nt sequence in database.

2.2 Genome database

After creating our custom gene database, we will create our custom genome database. In our case, as we are studying the *Pseudomonas putida* database) genomes, we will retrieve XX *Pseudomonas putida* database) genomes from the NCBI Pathogens database. In order to have a balanced representation of different isolation matrixes, we will take into account the environmental conditions of where each bacteria was sampled, and form our database with bacteria from different isolation environments. The columns present in this database will be the following (as all the strains will be from the *Pseudomonas putida* database) species, we will not have a column specifying the species):

- Strain: Name of the bacterial strain
- Assembly: Code for the genomic assembly of the strain.
- Isolation matrix: Matrix from where the bacteria was isolated. In our case we will have three: Soil, Water and Rhizomatic.
- Specific isolation matrix: To specify the type of isolation matrix. For example, if our general isolation matrix is soil, the specific isolation matrix could be Industrial, agricultural...
- More specific isolation matrix: More specification in order to have more concrete information. In the case of industrial soil, more specific isolation matrix could be coal waste, electronic waste...
- Host: If the bacteria was isolated from a host lie a plant or an animal.
- Pollution: If the isolation matrix was polluted in the moment of sampling, and what type of pollutant.
- More info: More additional info tat could be relevant for our study
- Date: the date of sampling
- Localization: the country of sampling
- Paper: A paper that registers the isolation.

After having both databases ready, we will outline the steps for performing a BLAST local search with both databases. Starting with some query sequences all joined in a multifasta file, we will conduct the local search of these sequences in several genomes. In our case, we will be working with 60 *Pseudomonas putida* database) genomes.

3 BLAST installation

First, we need to install BLAST+, a command-line tool from NCBI that allows us to run BLAST searches on our own computer. For this, using conda is usually recommended, as it has an easy installation and creates an environment with the necessary requirements. In our Linux terminal, we run the following code:

```
conda install -c bioconda blast
```

4 Prepare the genomes as databases

To compare our genes against each *Pseudomonas putida* genome, we first need to set up each genome as a BLAST database. This database format allows BLAST to search quickly and efficiently. First, place all genome FASTA files in a single directory (folder) for easy access. In the terminal, navigate to the folder containing your genome files. We will create another folder outside the FASTA folder to contain the obtained output databases first. We will call it Databases. Once this is done, run this command for each genome file inside the FASTA folder:

```
for genome in *.fasta; do makeblastdb -in "$genome" -dbtype nucl -out "../
    Databases/${genome%.fasta}_db"; done
```

This will create several files that correspond to the databases of each genome.

5 Prepare gene queries as multifasta

We have our gene database in an excel file, and now we have to create a multifasta file that contains the genomic sequences of all our genes along their corresponding FASTA headers. This multifasta file will be used as query in the BLAST local search.
In order to convert the excel file into a multifasta file, we will run the following code in Google Colab:

```
import pandas as pd

excel_path = "/content/drive/MyDrive/Libe/Gene_database.xlsx"
df = pd.read_excel(excel_path)
```

```

# Adjust these column names if they differ in the actual file
gene_name_col = "gene name"
description_col = "gene description"
species_col = "species"
sequence_col = "nt"

with open("/content/drive/MyDrive/Libe/BLAST_gene_count/genes.fasta", "w") as fasta_file:
    for index, row in df.iterrows():
        gene_name = row[gene_name_col]
        description = row[description_col]
        species = row[species_col]
        sequence = row[sequence_col]

        # Remove any whitespace in the sequence and split it into lines of 60 characters
        sequence = str(row[sequence_col]).replace(" ", "").replace("\n", "")
        sequence_lines = [sequence[i:i+60] for i in range(0, len(sequence), 60)]

        # Write the FASTA header and sequence lines
        fasta_file.write(f">{gene_name} | {description} | {species}\n")
        fasta_file.write("\n".join(sequence_lines) + "\n")
```

The code above is available in the following Google Colab link.

6 Run BLAST to search for genes in genome

Now that we have BLAST databases for each genome, we can search for our gene sequences across all genomes. First make sure all gene sequences are in a single FASTA file (e.g., genes.fasta), and that it is located inside the Databases folder. This file will act as the "query" in the BLAST search.
We will create a folder that will contain the BLAST hit results. This folder will be located outside the FASTA and Database folders, ant it will be called BLAST_hits.
Use the following command while inside the Databases folder to search for genes in each genome.

```
for db in *_db; do blastn -query genes.fasta -db "$db" -outfmt "7 qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue bitscore" -out "../BLAST_hits/${db}_blast_results.txt"; done
```

This will create multiple output files, each named like genome1_blast_results.txt, genome2_blast_results.txt, etc.

7 Order the results

In order to have all the results in one excel, we will manually pass all the BLAST hits from their file to a common file with the following code (run it while inside the BLAST_hits folder):

```
cat genome1_blast_results.txt | grep -vE "^#" > genome1_blast_results.txt
```

We will open this new file with the Excel application, and we will later upload it to drive. Once in drive, we will call the first tab 'Blast hits' and we will create a second tab that will be called 'Gene counts'. Now we want to count the number of times each gene appears in each strain, so we will use the following code in Google Colab to retrieve the gene counts from the recently created excel file:

```

# Load the two sheets from the single Excel file
file_path = "/content/Gene_counts.xlsx"
```

```
blast_hits = pd.read_excel(file_path, sheet_name="Blast Hits")  # Replace with the actual sheet name
gene_counts = pd.read_excel(file_path, sheet_name="Gene Counts")  # Replace with the actual sheet name
# Step 1: Count occurrences of each gene per strain
# Group by 'strain' and 'gene' and count occurrences
gene_occurrences = blast_hits.groupby(['strain', 'gene']).size().unstack(fill_value=0)

# Step 2: Prepare the Gene Counts DataFrame
# Create a new DataFrame to hold the counts
gene_counts_df = pd.DataFrame(index=gene_occurrences.index)
# Fill in the gene names from the columns of gene_occurrences
for gene in gene_occurrences.columns:
    gene_counts_df[gene] = gene_occurrences[gene].fillna(0)  # Fill counts, replacing NaN with 0
# Step 3: Reset index to have 'strain' as a column
gene_counts_df.reset_index(inplace=True)
# Step 4: Save the updated gene_counts_df back to the same Excel file
with pd.ExcelWriter(file_path, engine="openpyxl", mode="a", if_sheet_exists="replace") as writer:
    gene_counts_df.to_excel(writer, sheet_name="Gene Counts", index=False)
```

The code above is available in the following Google Colab link.