Intro to Bioinformatics - The flu

BIOS P-13532

2024-7-18

Bioinformatics: An Example in Influenza

Let's get started! We'll use influenza sequence data from the Bacterial and Viral Bioinformatics Resource Center (BV-BRC). Download the files "BVBRC_genome_sequence.fasta" and "BVBRC_genome.csv" from the Canvas page. Contained in these files are the sequences and metadata for 200 strains of influenza (ie. the flu.)

For simplicity, let's make a folder on our desktop called "omics_informatics" and put all our files (the markdown file, the metadata file, and the fasta file) there. Now we need to tell R where this is. - Mac users can find this by right clicking on the file name in finder, holding down the option key, and clicking "copy FILENAME as path name" - Windows users can find this by selecting (ie clicking once on) the file and then pressing ctrl+shift+c

```
setwd("/Users/alexandria/Desktop/omics_informatics")
```

Complete the below code block to set up our environment in R and load the data files you've just downloaded into R.

```
# First we will install the BiocManager so that we can
# easily install packages from Bioconductor in our R
# environment.  If prompted with 'update all/some/none?',
# enter a If prompted with 'Do you want to install from
# sources the packages which need compilation?' enter n
if (!requireNamespace("BiocManager", quietly = TRUE)) install.packages("BiocManager")
# We will then use BiocManager to install the remaining
# packages needed for this tutorial
BiocManager::install(c("msa", "tidyverse", "Biostrings", "ape",
    "seqinr", "ggtree", "RColorBrewer", "ggmsa"))
# Once they are installed the libraries can be loaded into
# R.  In the future, if you have already installed a
# program on your computer as we did above, you can simply
# proceed for the library(package) commands
library(msa)
library(tidyverse)
library(Biostrings)
library(ape)
library(seqinr)
library(ggtree)
library(RColorBrewer)
library(ggmsa)

# Excellent! The Rstudio environment is set up and we're
# ready to go!

# We will now create variables for the names of the
```

```
# sequence and metadata in your computer so R can find this
# later.
fasta_file <- "BVBRC_genome_sequence.fasta"
metadata_file <- "BVBRC_genome.csv"

# Now that we have those variables we can load the data as
# follows:
sequences <- readDNAStringSet(fasta_file)

# For the metadata information we will assign the GenBank
# Accession number column as the row names (the identifiers
# for the row data) to coordinate with the ids in our
# sequence data
metadata_df <- read_csv(metadata_file) %>%
    column_to_rownames("GenBank_Accessions") %>%
    as.data.frame()

# Let's take a look and see what our data looks like so
# far.

# If we type the name of the variable for our sequences,
# Rstudio will print them to the console below and we can
# see a shortened version of our sequences
sequences
```

```
## DNAStringSet object of length 200:
##       width seq                                               names
##   [1]  1701 ATGAAGGCAGTACTAGTAGTCC...ACAGTGCAGAATATGTATTTAA MT277374
##   [2]  1701 ATGAAGGCAGTACTAGTAGTCC...ACAGTGCAGAATATGTATTTAA MT277463
##   [3]  1698 ATGAAAGTAAAACTAATGGTTC...ACAGTGTAGAATATGCATCTAA MT277487
##   [4]  1701 ATGAAGGCAATACTAGTAGTCT...ACAGTGCAGAATATGTATTTAA MT367647
##   [5]  1701 ATGAAGGCAATACTAGTAGTCT...ACAGTGCAGAATATGTATTTAA MT372514
##   ...   ... ...
## [196]  1701 ATGAAGGCAGTAATAGTAGTCT...ACAGTGCAGAATATGTATCTAA MT052164
## [197]  1701 ATGAAGGCAGTACTAGTAGTCC...ACAGTGCAGAATATGCATTTAA MT154189
## [198]  1698 ATGAAAGTAAAACTACTGATCC...GCAATGCAGAATATGCATCTGA MT154193
## [199]  1701 ATGAAGGCAATACTAGTAGTTC...ACAGTGTAGAATATGTATTTAA MT232793
## [200]  1701 ATGAAGGCAATACTAGTAGTTC...ACAATGTAGAATATGTATTTAA MT233093
```

```
# If we want to see the raw sequence data we can type
view(sequences)
# If we want to investigate the set of our DNA FASTA
# sequences we can tell R to open a new tab with the
# information using
View(sequences)
```

```
## Error in check_for_XQuartz(file.path(R.home("modules"), "R_de.so")): X11 library is missing: install
```

```
# Run the same commands for metadata and see what
# information you find
```

Using the seperate table that the View(metadata_df) command yields, look at some of the details the metadata contains. What is the host? Where did the strains originate? How many sequences are contained in our sample? Are these all the same type of flu or do they differ?

Once we get a handle on what we're working with, we can proceed to align the flu sequences and start asking

some questions.

```
# Assign the metadata to the sequence data so they're
# associated
metadata(sequences) <- metadata_df[match(names(sequences), rownames(metadata_df)),
    ]
# Now run View(sequences) again. What has changed?
View(sequences)
```

```
## Error in check_for_XQuartz(file.path(R.home("modules"), "R_de.so")): X11 library is missing: install
```

```
# If you'd like to view the metadata for a specific
# sequence you can type
# sequences@metadata['ACCESSION_NUMBER',] for example

sequences@metadata["OK047469", ]
```

```
##                                  Strain Isolation_Country Host Collection_Date
## OK047469 A/swine/Missouri/A02635993/2021               USA  Pig         7/27/21
##          Segment Subtype
## OK047469       4    H3N2
```

```
# Now that we've combined our data let's make an alignment.
# Let's see what our options for aligning these sequences
# are by pulling up the help panel for our aligner.
`?`(msa)

# What different options do we have for this alignment
# algorithm? Do you think we need to specify them all?

# Given the time constraints, we'll run an alignment using
# the Muscle algorithm and assign it to a variable called
# 'alignment'. Below this code block is information about
# the different alignment algorithms; why msa runs, take
# some time to read about the differences.
alignment <- msa(sequences, method = "Muscle")
```

The MUSCLE and Clustal Omega aligner information below is from Geneious, a commonly used sequence aligner. Information about this and other aligners used by Geneious can be found here. The ClustalW aligner information is from Mega.

MUSCLE

MUSCLE is is a progressive aligner that features rapid sequence distance estimation using k-mer counting, progressive alignment using a profile function termed the log-expectation score, and refinement using tree-dependent restricted partitioning of the sequences. The algorithm is described at http://nar.oxfordjournals.org/content/32/5/1792.full.pdf+html and a full manual is available at http://www.drive5.com/muscle/manual/index.html.

Suitable for medium-large alignments up to 1000 sequences. Not suitable for sequences with low homology N-terminal and C-terminal extensions.

Clustal Omega

Clustal Omega is a fast, accurate aligner suitable for alignments of any size. It uses mBed guide trees and pair HMM-based algorithm which improves sensitivity and alignment quality. A full description of the algorithms used by Clustal Omega is available in the Molecular Systems Biology paper Fast, scalable generation of

high-quality protein multiple sequence alignments using Clustal Omega. Latest additions to Clustal Omega are described in Clustal Omega for making accurate alignments of many protein sciences. See also the Clustal Omega website for further details.

Suitable for very large datasets of over 2000 sequences Multi-threaded for faster alignment. Suitable for sequences with long, low homology N-terminal or C-terminal extensions Not suitable for alignment of sequences with large internal indels

ClustalW

ClustalW is a widely used system for aligning any number of homologous nucleotide or protein sequences. For multi-sequence alignments, ClustalW uses progressive alignment methods. In these, the most similar sequences, that is, those with the best alignment score are aligned first. Then progressively more distant groups of sequences are aligned until a global alignment is obtained. This heuristic approach is necessary because finding the global optimal solution is prohibitive in both memory and time requirements. ClustalW performs very well in practice. The algorithm starts by computing a rough distance matrix between each pair of sequences based on pairwise sequence alignment scores. These scores are computed using the pairwise alignment parameters for DNA and protein sequences. Next, the algorithm uses the neighbor-joining method with midpoint rooting to create a guide tree, which is used to generate a global alignment. The guide tree serves as a rough template for clades that tend to share insertion and deletion features. This generally provides a close-to-optimal result, especially when the data set contains sequences with varied degrees of divergence, so the guide tree is less sensitive to noise.

```
# Once the alignment has finished running we can take a
# peak at our data.  We can check what type of data that
# variable contains
class(alignment)
```

```
## [1] "MsaDNAMultipleAlignment"
## attr(,"package")
## [1] "msa"
```

```
is(alignment)
```

```
## [1] "MsaDNAMultipleAlignment" "DNAMultipleAlignment"
## [3] "MsaMetaData"             "MultipleAlignment"
```

```
# We can view the direct output from the msa alignment but
# it ay not be very interesting given the amount of data
# and our screen size.
alignment
```

```
## MUSCLE 3.8.31
##
## Call:
##    msa(sequences, method = "Muscle")
##
## MsaDNAMultipleAlignment with 200 rows and 1850 columns
##       aln                                                  names
##   [1] ------------------------...------------------------ MZ663745
##   [2] ------------------------...------------------------ OK143188
##   [3] ------------------------...------------------------ OK384253
##   [4] ------------------------...------------------------ MZ606843
##   [5] ------------------------...------------------------ OL331236
##   [6] ------------------------...------------------------ OK384247
##   [7] ------------------------...------------------------ OK047469
##   [8] ------------------------...------------------------ OK625349
```

```
##   [9] ------------------------...------------------------ MZ502593
##   ... ...
## [193] ------------------------...------------------------ MT982667
## [194] ------------------------...------------------------ OK064050
## [195] ------------------------...------------------------ MW776849
## [196] ------------------------...------------------------ OM935963
## [197] ------------------------...------------------------ MZ485426
## [198] ------------------------...------------------------ MW320421
## [199] ------------------------...------------------------ MW075260
## [200] ------------------------...------------------------ OM935917
##   Con ------------------------...------------------------ Consensus
```

```
# If we'd like to view more we can use the print the full
# alignment using print(alignment, show='complete')

# What do you notice when we do this? How do you even being
# to sort this into meaningful information?  Let's extract
# the alignment data from our alignment object and save it
# to a fasta file as a backup.

alignment_clean <- unmasked(alignment)
writeXStringSet(alignment_clean, "alignment.fasta", append = FALSE,
    compress = FALSE, compression_level = NA, format = "fasta")

# How does this look now?
alignment_clean
```

```
## DNAStringSet object of length 200:
##       width seq                                             names
##   [1]  1850 ------------------------...------------------------ MZ663745
##   [2]  1850 ------------------------...------------------------ OK143188
##   [3]  1850 ------------------------...------------------------ OK384253
##   [4]  1850 ------------------------...------------------------ MZ606843
##   [5]  1850 ------------------------...------------------------ OL331236
##   ...   ... ...
## [196]  1850 ------------------------...------------------------ OM935963
## [197]  1850 ------------------------...------------------------ MZ485426
## [198]  1850 ------------------------...------------------------ MW320421
## [199]  1850 ------------------------...------------------------ MW075260
## [200]  1850 ------------------------...------------------------ OM935917
```

```
# Let's visualize this a bit more nicely. This is a lot of
# data so this may take a few minutes!

png("alignment.png", width = 2000, height = 6000)
ggmsa(alignment_clean, 300, 350, color = "Clustal", font = "DroidSansMono",
    char_width = 0.5, seq_name = TRUE)
dev.off()
```

```
## pdf
##   2
```

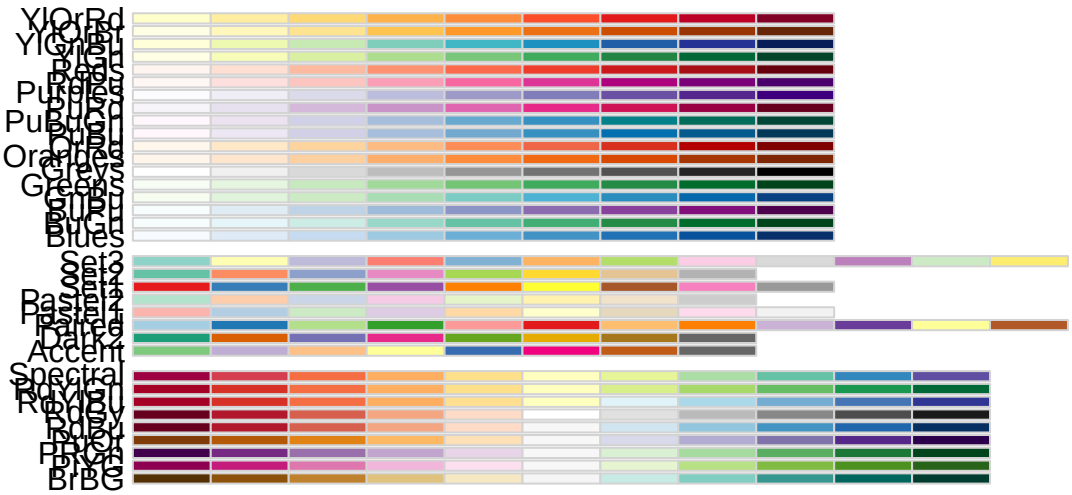
Relationships of Influenza Strains

Now that we have an alignment of our sequences, we'll build a phylogenetic tree to look at how they're related and which strains likely arose from common ancestors.

```
# To build a tree we need to quantify how different our
# sequences are from each other. To do this. we calculate a
# **distance matrix**. This calculates a distance value for
# each pair of sequences and saves them into a Q matrix.
dist_matrix <- as.DNAbin(alignment_clean) %>%
    dist.dna()

# Once we've calculated these values, we can build a tree.
# There are many models for how best to build phylogenetic
# trees, but this topic can (and does!) take whole classes
# to cover in depth. For the purpose of this tutorial,
# we'll build a neighbor joining tree. Neighbor Joining is
# one method of tree building which works by iteratively
# joining the most closely related pairs of species or
# sequences, creating a tree where the distances between
# the branches represent the genetic or evolutionary
# differences. This approach allows us to visualize and
# understand the evolutionary connections among different
# organisms or genetic sequences.

tree <- nj(dist_matrix)

# Now that we've built a tree, let's create a png of the
# tree and take a look! We'll plot the tree and also load
# our metadata in so we can color by different information.

# To use our metadata we need a column called 'taxa' with
# the accession numbers. To do this we'll run
metadata_df <- rownames_to_column(metadata_df, var = "taxa")

# You can change the 'Isolation_Country' column name in '
# aes(color=Isolation_Country) ' to a different column name
# color by different data traits. To view columnnames enter
colnames(metadata_df)
```

```
## [1] "taxa"              "Strain"            "Isolation_Country"
## [4] "Host"              "Collection_Date"   "Segment"
## [7] "Subtype"
```

```
# You can change the color to a different pallet by
# changing the 'Set3' to a different colorbrewer set. to
# see the options type
display.brewer.all()
```




```
png("tree.png", width = 2000, height = 2000)
ggtree(phylo_tree) %<+% metadata_df + geom_tiplab(aes(color = Isolation_Country)) +
    scale_color_brewer(palette = "Set2")
```

```
## Error in eval(expr, envir, enclos): object 'phylo_tree' not found
```

```
dev.off()
```

```
## pdf
##   2
```

```
# Try different variables. Do any of them seem to explain
# the groupings?  Which group is the most diverged (ie. has
# the longest branch) from the other groups? What do you
# think that means?
```

SNPs all around

Let's look a bit deeper at what's causing these differences. Here we'll investingate the SNPs between these sequences.

```
# Convert the alignment to a matrix format
alignment_matrix <- as.matrix(alignment_clean)

# Create a dataframe to store SNP information
snp_df <- data.frame(Position = 1:ncol(alignment_matrix), SNPs = character(ncol(alignment_matrix)),
    stringsAsFactors = FALSE)

# Loop through each position in the alignment
for (i in 1:ncol(alignment_matrix)) {
    # Extract the nucleotides at the current position
    nucleotides <- unique(alignment_matrix[, i])

    # Check if it is a SNP (more than one unique
    # nucleotide)
    if (length(nucleotides) > 1) {
        # Store the SNPs in the dataframe
        snp_df$SNPs[i] <- paste(nucleotides, collapse = "/")
    }
}
```

```r
# Remove positions without SNPs
snp_df <- snp_df %>%
    filter(!is.na(SNPs))

# To view all of your SNPs you can print the dataframe by
# typing print(snp_df)

# It may be easier to visualize this over the consensus
# sequence so we can see where in the overall alignment
# these SNPs are occurring.

# First we need to extract the consensus sequence from our
# original multisequence alignment

consensus <- msaConsensusSequence(alignment) %>%
    as.character()

# Then we create a dataframe for printing
print_df <- data.frame(Position = snp_df$Position, Consensus = unlist(strsplit(consensus,
    "")), SNP_Alternate = snp_df$SNPs, stringsAsFactors = FALSE)

# And finally we create a loop to print the position in the
# first column, the consensus sequence base in the second
# column and the SNPs in parentheses the 3rd column Print
# the SNP analysis Define the output file path and name
output_file <- "snp_analysis.tsv"

# Open the output file for writing
file_conn <- file(output_file, "w")

# Loop through each row of the dataframe and write to the
# output file
for (i in 1:nrow(print_df)) {
    position <- print_df$Position[i]
    consensus_base <- print_df$Consensus[i]
    snp_alternate <- print_df$SNP_Alternate[i]

    # Write the SNP analysis to the file
    cat(position, "\t", consensus_base, "\t", "(", snp_alternate,
        ")", "\n", file = file_conn)
}

# Close the output file connection
close(file_conn)

# Go to your working directory. You can now open this file
# in a plain text editor or excel to view.
```