Comparative RNA-Seq: Signal to Noise Documentation

T.J Nicholson & P.P Gardner

March 2022

Contents

Data Selection

Strains and experiments

How to get the summary table for the genera for the RNA-seq data:

- Search the SRA section of ncbi with the genus name and [‘Organism’] tag.
- Select “Send results to Run selector”
- Select the “Metadata” button of the “Total” row and “Download” Column.
- The genome accession for each added genome was looked up on NCBI using the strain name in the file or by search for the given experiment in the SRA section.
- Create a folder for the genus e.g. `mkdir Escherichia`
- Create a folder using the GenBank assembly accession e.g. `mkdir GCA_002504285.1.data/`
- Create an `experiments_list.txt` file listing all of the SRA experiments for the given genome and save the file in the `genus` folder.
- Run `callPeaksforGenome.sh -g <GCA Accession> -s` from within the `genus` folder.
- List of representative genomes selected from RefSeq95
  - 2 genomes from each genera were selected (where possible)

Predicting sRNAs

- The `callPeaksforGenome.sh` carried out a number of steps
  - `fasterq-dump <SRA Accession>` downloaded each of the RNA-Seq files listed in the `experiments_list.txt` file.
  - `fetch_genomes_from_GCA.sh` downloaded the nucleotide FASTA file and the annotation file in GFF3 format.
  - Known ncRNAs are identified using `cmscan` with models from Rfam
  - `sra2plot.1.0.3.sh` took these files and produced a plot file (showing read depths at each nucleotide position) for each of the experiments.
  - `run_rnaPeakCalling.R` took these plot files and identfied regions of RNA expression tht occured in intergenic regions
  - `combine_gff_files.R` took the expressed regions from each of the individual files (including the annotation file) and merged regions where there was an overlap.
- The resulting regions are separated into `Known sRNA` and `Putative RUFs` based on whether an expressed region overlapped with a previously identified ncRNA
- The negative control group (RINCs) was generated using `get_random_srna_sequences.py`
  - For each putative RUF, a region of the same size was selected at random from an intergenic region in the same genome

```
for file in *.txt; do
  accession=`basename $file _new_calls.txt`;
  get_random_srna_sequences.py -a $accession;
  done
```

- The nucleotide sequences for each group were obtained from the nucleotide FASTA files using the coordinates of the regions

Analysis

Homology Search

- `run_sRNA_nhmmer.sh -d <group_seqs.fna> -f <path/to/sequences/> -e fna`
  - The `-e` flag is for the the expected extension of the input files.
  - This loops through all the individual sequences in each data set and carries out a similarity search against all the sequences in the data set.
  - The matched sequences are removed from the remaining search list to prevent multiple identical alignments being produced.
- `run_sRNA_nhmmer.sh -d <path/to/genomes/> -f <path/to/initial_alignment> -o <path_to_output>/ -e stk -E 1e-3`
  - The `-E` flag sets the e-value threshold for inclusion.
  - This gets repeated two times to generate deep alignments for the sequences from each data set.
- A check was carried out for each of the groups (known sRNAs, predicted RUFs and RINCs) that none of the alignments shared sequences.
  - RINCs sharing sequences with either of the other data sets were removed.
  - Predicted RUFs sharing sequences with the known sRNAs were removed.

```
group='predicted' #repeat for each group
cd ${group}/alignments/
> ../${group}_contig_pos.txt
for file in *.stk;
do
  ID=`basename $file .stk`

  # extract the contig name and position for each sequence in each stockholm
  #alignment file
  grep "/" $file | grep -v "//" | cut -d ' ' -f1 | sed 's/\// /g' |
  sed 's/-/ /g'| cut -d '|' -f2 | sort | uniq | sed -e "s/$/ $ID/" >>
  ../${group}_contig_pos.txt
done
```

```
library(GenomicRanges)
ncdat <- read.table("negative_control_contig_pos.txt", sep = " ", fill = T)

pcdat <- read.table("positive_control_contig_pos.txt", fill = T)

preddat <- read.table("predicted_contig_pos.txt", fill = T)

pcdat <- reformatContigPositionData(dat = pcdat)
ncdat <- reformatContigPositionData(dat = ncdat)
preddat <- reformatContigPositionData(dat = preddat)

nc.pc.Dat <- getOverlapIDs(queryDat =ncdat, targetDat = pcdat)
nc.pred.Dat <- getOverlapIDs(queryDat =ncdat, targetDat = preddat)
pc.pred.Dat <- getOverlapIDs(queryDat =preddat, targetDat = pcdat)

dat <- data.frame(ids = c(nc.pc.Dat$id1, nc.pred.Dat$id1))

write.table(dat, file = "not_negative_control_ids.txt", row.names = F, col.names = F, quote = F)

dat2 <- data.frame(ids =  pc.pred.Dat$id1)

write.table(dat2, file = "not_predicted_ids.txt", row.names = F, col.names = F, quote = F)
```

- A similar check was carried out within each group to reduce redundancy
  - Alignments sharing a sequence were merged into a single alignment.

```
#read in the data
dat <- read.table("predicted_contig_pos.txt", sep = " ", fill = T, as.is = T)
dat <- dat %>% select(V1, V4)
#split out the columns of interest
dat <- dat %>% separate(col = V1, into = c("contig", "coordinates"), sep = "\\/", remove = F)
dat <- dat %>% separate(col = coordinates, into = c("start", "stop"), sep = "-", remove = F)
dat <- dat %>% dplyr::rename(srna = V4) %>% select(contig, srna, start, stop) %>% mutate(strand = "+")
colnames(dat) <- c("contig","srna", "start", "stop", "strand")

#rearrange the start and stop so th at that the stop > start
dat <- dat %>% mutate(tmpstart = ifelse(start < stop, start, stop),
                      tmpend = ifelse(start > stop, start, stop))
```

```
#Set up the data to be used in genomic ranges overlap search
query <- GRanges(seqnames = dat$contig,
                 ranges = IRanges(start = dat$tmpstart, end = dat$tmpend),
                  strand = dat$strand, query_name = dat$srna)
lookup1 <- data.frame(id1 = dat$srna, queryHits = c(1:length(dat$srna)))
lookup2 <- data.frame(id2 = dat$srna, subjectHits = c(1:length(dat$srna)))


#Run search for overlaps and merge with original input to obtain pairs of
#overlaps
tmp <- GenomicRanges::findOverlaps(query, query, type = 'any')
tmp <- as.data.frame(tmp)
tmp <- tmp %>% left_join(lookup1) %>% left_join(lookup2)
tmp <- tmp %>% filter(id1 != id2)
smallDat <- tmp %>% select(id1, id2) %>% unique()

#Swap columns and combine so that every combination is present in both columns
s2 <- smallDat
s2$id1 <- smallDat$id2
s2$id2 <- smallDat$id1
smallDat <- smallDat %>% bind_rows(s2) %>% unique()

ids <- unique(dat$srna)

#Loop over the ids and run a check to see if there are any overlapping ids
#to speed this up check if an id has already been seen and skip it.
#Write a list of overlapping ids to a unqiue file
checked_ids <- c()
for(item in ids){
  if(item %in% checked_ids){
    next
  }
  current_ids <- groupOverlapItems(smallDat, item, current_ids = c())
  checked_ids <- c(checked_ids, current_ids)
    write.table(x = current_ids,
                file = paste("combined_alignment_ids/",
                             current_ids[1],
                             "_combined_list.txt", sep = ""),
                append = T, quote = F, row.names = F, col.names = F)
}
```

```
#move into the folder where the lists of overlapping ids are kept
cd combined_alignments_ids/
mkdir -p fasta
mkdir -p hmm
mkdir -p alignments

#loop through the files that contain overlapping ids
for file in *.txt;
do
max_num="0"
outname=`basename $file _combined_list.txt`
#Check if there is only one ID listed in a file. If so there is no merge
```

```
#required.
line_count=`wc -l $file | cut -d ' ' -f1`
if (( $line_count == 1 ));then
cp ../large_alignments/alignments/$outname.stk ./alignments
continue
fi

#loop through the file and extract the sequences from each of the stk files
#that correspond to the ids that are overlapping
> fasta/$outname.fa
while read line;
do
esl-reformat fasta ../large_alignments/alignments/$line.stk >> fasta/$outname.fa
current_num=`esl-alistat ../large_alignments/alignments/$line.stk | grep "Number of sequences" | cut -d

#check if the current alignment is the longest
if(( $current_num > $max_num ));
then
max_seq="$line.stk"
max_num="$current_num"
fi
done < $file

#build an hmm using the longest alignment
hmmbuild hmm/$outname.hmm ../large_alignments/alignments/$max_seq

#build an alignment using all the sequences from the overlapping ids
hmmalign --informat fasta hmm/$outname.hmm fasta/$outname.fa | esl-alimask -g --gapthresh 0.8 -p --pfrac
done
```

File setup

- There are some files that are consistently used throughout the analysis
- genome_contig_pairs.txt is a file that maps the genome GCA_ ids to the contig NC ids. - this is necessary due to the alignment reporting contig names, but the predicted RUFs belonging to genomes rather than contigs.
- query_target_pairs.txt consists of each alignment ID and the contig IDs of each sequence within the alignment.

```
#loop through all the genomes in the analysis
cd representative_genomes/
> ../genome_contig_pairs.txt
for file in GC*.fna;
do
  ID=`echo $file | cut -d '.' -f1,2 | cut -d "_" -f1,2`;
  grep ^">" $file | cut -d ' ' -f1 | sort | uniq | sed 's/>//g' | sed -e "s/$/   $ID/" >> ../genome_con
done

#for each group loop through all the alignments and extract the contig IDs
#of all the sequences within each alignment
group="positive_control"
cd ${group}/alignments
> ../query_target_pairs.txt
```

```
for file in *.stk;
do
  ID=`echo $file | cut -d '.' -f1,2 | cut -d "_" -f1,2`;
  ID_2=`echo $file | cut -d '.' -f1,2 `;
  grep ^"#=GS" $file | sort | uniq | cut -d "/" -f1 | cut -d ' ' -f2 | rev | cut -d '|' -f1 | rev | sed
done
```

Evolutionary Distance

- A `cmscan` of all the genomes using the RF00177 Rfam model (bacterial ribosomal SSU) was carried out.
- Phylip was used to build a maximum likelihood phyogenetic tree.
  - The `RF00177.stk` alignment was reformatted to `.phylip` format
  - `esl-reformat phylip RF00177.stk > RF00177.phylip`
  - This was used as the input for `dnadist` with an output of `RF00177.dnadist`.
- The file was not formatted in a usable way for reading into R
  - `cat  RF00177.dnadist | tr '\n' ' ' | sed 's/ N/\nN/g' > RF00177.dists`

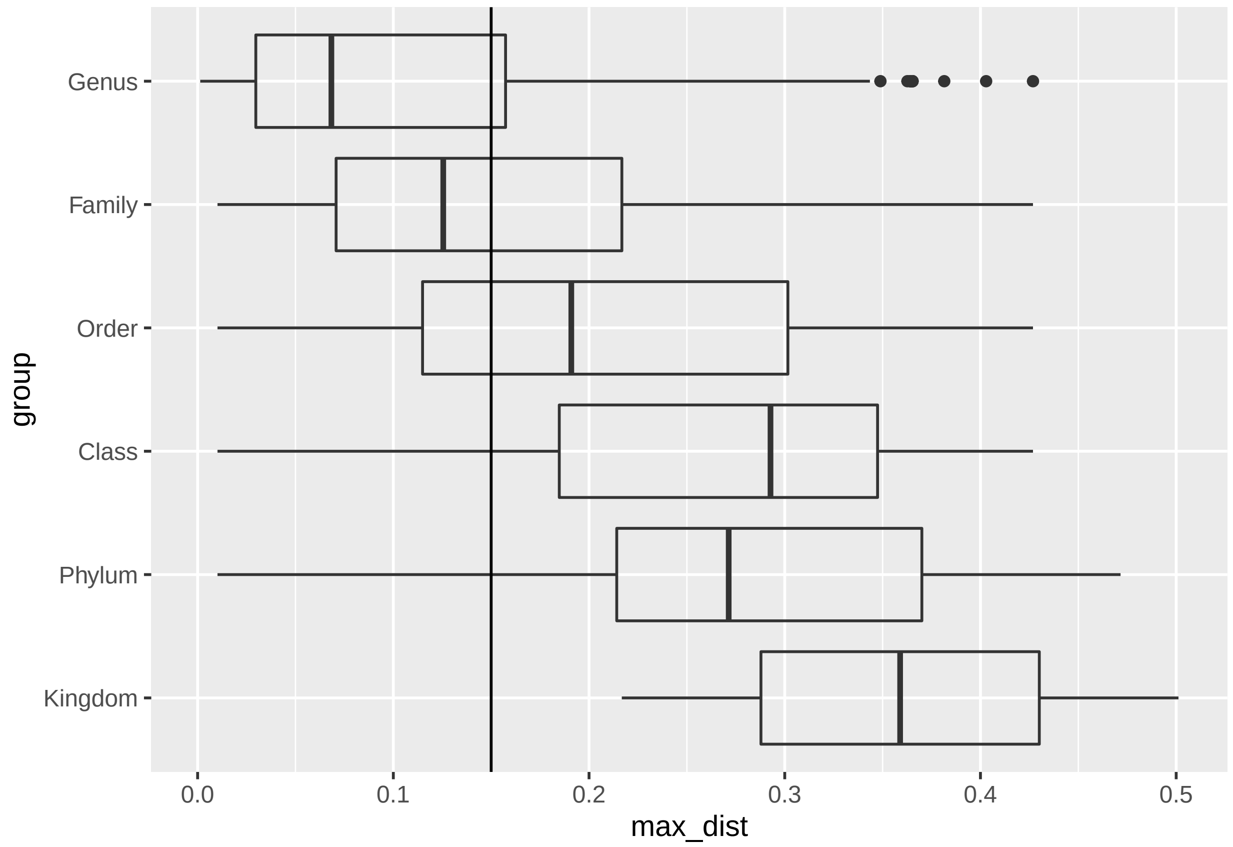
```
#open the dnadist file
dat <- read.table("RF00177.dists", sep = "",
                  header = F, fill = T, stringsAsFactors = F, as.is = T)
dat <- dat %>% filter(!is.na(V2))
colnames(dat)[1] <- "phylip_id"
colnames(dat) <- c("names", dat$phylip_id)

#reformat the data to use in subsequent steps
meltDat <- melt(dat)
colnames(meltDat) <- c("phylip_id", "query.name", "distance")

#open list of phylip ids (truncated) and contig ids
load(file = "current_ids.Rda")
current_ids <- current_ids %>% unique()

#open list of genome and contig ids
contig_labels <- read.table("genome_contig_pairs.txt")
colnames(contig_labels) <- c("id", "target.genome")

#combine the phylip ids with the genome ids
contig_labels <- contig_labels %>%
  left_join(current_ids, by = "id") %>%
  select(-b)

#combine the genome ids with the distances data for the first column
colnames(contig_labels) <- c("target.id", "target.genome", "phylip_id")
meltDat <- meltDat %>% left_join(contig_labels, by = 'phylip_id')

#combine the genome ids with the distances data for the second column
colnames(meltDat)[1:2] <- c("target.name", "phylip_id")
colnames(contig_labels) <- c("query.id", "query.genome", "phylip_id")
meltDat <- meltDat %>% left_join(contig_labels, by = 'phylip_id')
colnames(meltDat)[2] <- c("query.name")
```

```
#take the pairs of genomes and arrange based on distances (to account for
#multiple contigs). Sort these and select a single distance for each pair.
meltDat <- meltDat %>%  group_by(target.genome, query.genome) %>% arrange(as.numeric(distance)) %>% muta
save(meltDat, file = "~/bin/R/r_files/distanceMelt.Rda")

#Reshape the data back to a matrix (not sure this was needed but helped for
#visualising the problem)
mat <- reshape2::acast(data = meltDat %>%
                         select(target.genome, query.genome, distance),
                       formula = target.genome ~ query.genome)
distanceMat <- as.data.frame(mat)
save(distanceMat, file = "~/phd/RNASeq/distanceMat.Rda")

#include a save a load point as the previous steps were time consuming and
#this allows for troubleshooting the next steps
load("~/phd/RNASeq/distanceMat.Rda")

#get the maximum distances between pairs of sequences for each alignment.
max_dists_pred <- calculateMaximumDistances('predicted')
save(max_dists_pred, file = "~/bin/R/r_files/max_dists_pred.Rda")
max_dists_pc <- calculateMaximumDistances('pc')
save(max_dists_pc, file = "~/bin/R/r_files/max_dists_pc.Rda")
max_dists_nc <- calculateMaximumDistances('negative_control')
save(max_dists_nc, file = "~/bin/R/r_files/max_dists_nc.Rda")
```

Data sets were filtered by distance < 0.15.

- Threshold was selected based on the distance between genomes within a family
  - Filtered at this stage to reduce computational time

```
load("~/bin/manuscript/data/allDists.Rda")
maxsGenus <- allDists %>% group_by(Genus.x, Genus.y) %>%
  summarise(max_dist = max(distance)) %>%
  mutate(same.taxa = ifelse(Genus.x == Genus.y, T, F))
maxsFamily <- allDists %>% group_by(Family.x, Family.y) %>%
  summarise(max_dist = max(distance)) %>%
  mutate(same.taxa = ifelse(Family.x == Family.y, T, F))
maxsOrder <- allDists %>% group_by(Order.x, Order.y) %>%
  summarise(max_dist = max(distance)) %>%
  mutate(same.taxa = ifelse(Order.x == Order.y, T, F))
maxsClass <- allDists %>% group_by(Class.x, Class.y) %>%
  summarise(max_dist = max(distance)) %>%
  mutate(same.taxa = ifelse(Class.x == Class.y, T, F))
maxsPhylum <- allDists %>% group_by(Phylum.x, Phylum.y) %>%
  summarise(max_dist = max(distance)) %>%
  mutate(same.taxa = ifelse(Phylum.x == Phylum.y, T, F))
maxsKingdom <- allDists %>% group_by(Kingdom.x, Kingdom.y) %>%
  summarise(max_dist = max(distance)) %>%
  mutate(same.taxa = ifelse(Kingdom.x == Kingdom.y, T, F))
genus1 <- maxsGenus %>% ungroup() %>% filter(same.taxa, max_dist > 0) %>% select(max_dist) %>% mutate(g
family1 <- maxsFamily %>% ungroup() %>% filter(same.taxa, max_dist > 0) %>% select(max_dist) %>% mutate
order1 <- maxsOrder %>% ungroup() %>% filter(same.taxa, max_dist > 0) %>% select(max_dist) %>% mutate(g
class1 <- maxsClass %>% ungroup() %>% filter(same.taxa, max_dist > 0) %>% select(max_dist) %>% mutate(g
phylum1 <- maxsPhylum %>% ungroup() %>% filter(same.taxa, max_dist > 0) %>% select(max_dist) %>% mutate
kingdom1 <- maxsKingdom %>% ungroup() %>% filter(same.taxa, max_dist > 0) %>% select(max_dist) %>% muta

maxs1 <- genus1 %>% bind_rows(family1, order1, class1, phylum1, kingdom1)
maxs1$group <- factor(maxs1$group, levels = c("Kingdom", "Phylum", "Class", "Order", "Family", "Genus")

p <- ggplot() +
geom_boxplot(data = maxs1 , aes(x = group, y = max_dist), fill = NA) + coord_flip() +
  geom_hline(yintercept = 0.15)
p
```

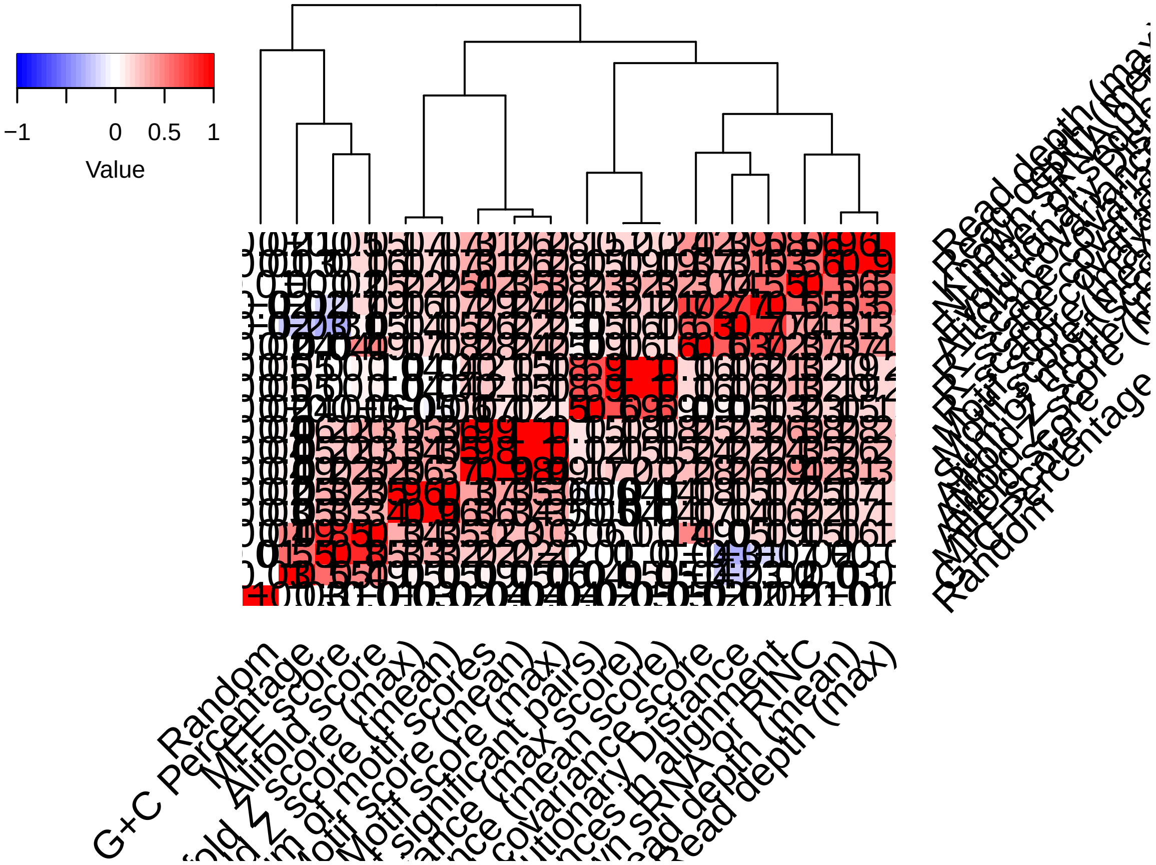


Correlation heatmap

A spearman correlation matrix was produced comparing multiple metrics for each feature. The best performing metrics (measured by comparing correlation with known sRNAs were kept).

```
load("~/bin/PhD/Chapter_4/chapter_4_files/randomForestDat.Rda")

dat <- dat %>% select(-ID, -power, -motif_count, -read.counts) %>% unique() %>%
  mutate(cov.min.eval = -log(cov.min.eval),
         z_mean = -z_mean,
         z_max = -z_max,
         alifold.score  =-alifold.score,
         mfe.score = -mfe.score,
         alifold_cov_score = -alifold_cov_score)

set.seed(101)
matNames <- c("MFE score", "G+C Percentage", "Evolutionary Distance",
              "Read depth (max)", "Read depth (mean)",
                        "R-scape covariance (mean score)",
              "R-scape covariance (max score)",
              "R-scape covariance (number of significant pairs)",
              "Motif score (mean)", "Motif score (max)",
              "Alifold Z score (mean)", "Alifold Z score (max)",
              "Alifold covariance score", "Alifold score",
              "Number of sequences in alignment", "Sum of motif scores",
              "Random", "Known sRNA or RINC")

matrixList <- calculateCorrelationMatrix(dat, matNames)
```

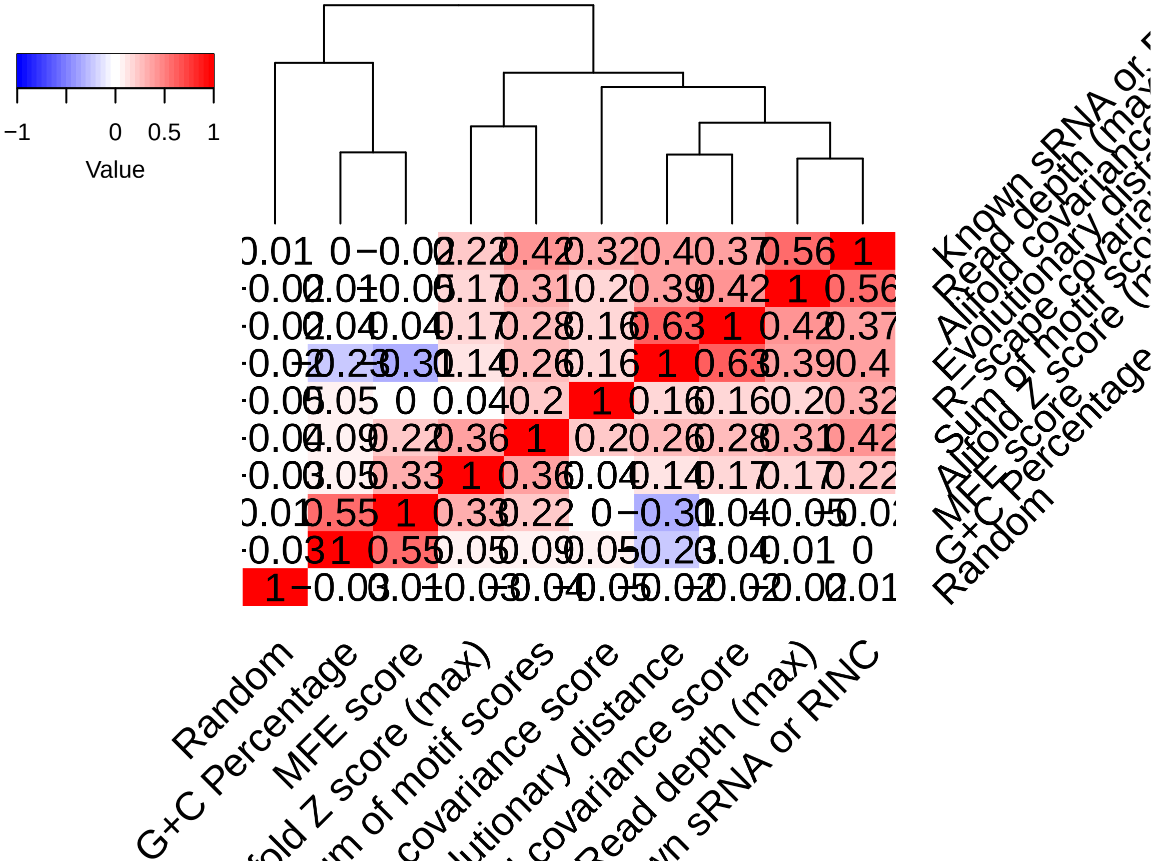
```
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

rhoMatrix <- matrixList$rhoMatrix
sigMatrix <- matrixList$sigMatrix

heatmap.2(rhoMatrix, cellnote=round(rhoMatrix, digits = 2),
          notecex=1.5,notecol="black", col=rev(redblue(40)),
          density.info="none", trace="none", dendrogram=c("column"),
          symm=F,symkey=T,symbreaks=T, scale="none", key.title = "",
          srtRow=45, adjRow=c(0, 1), srtCol=45, adjCol=c(1,1),
          breaks=(-20:20)/20, margins = c(8, 8), cexRow=1.5,
          cexCol=1.5,font=1)
```



```
if(FALSE){
svglite::svglite(paste0(figure_path, "/heatmap.svg"))
heatmap.2(rhoMatrix, cellnote=sigMatrix,notecex=1.5,notecol="black",
          col=rev(redblue(40)), density.info="none", trace="none",
          dendrogram=c("column"), symm=F,symkey=T,symbreaks=T, scale="none",
          key.title = "", srtRow=45, adjRow=c(0, 1), srtCol=45, adjCol=c(1,1),
          breaks=(-20:20)/20, margins = c(8, 8), cexRow=1.5, cexCol=1.5,font=2)

dev.off()
}


load("~/bin/PhD/Chapter_4/chapter_4_files/randomForestDat.Rda")

dat <- dat %>% select(group,
                      read.max.score,
                      distance,
                      alifold_cov_score,
                      cov.min.eval,
                      motif.sum,
                      z_max,
                      mfe.score,
                      gc.score) %>%
  unique() %>%
  mutate(cov.min.eval = -log(cov.min.eval),
         z_max = -z_max,
         mfe.score = -mfe.score,
         alifold_cov_score = -alifold_cov_score)

set.seed(101)
matNames <- c("Read depth (max)", "Evolutionary distance",
              "Alifold covariance score", "R-scape covariance score",
              "Sum of motif scores", "Alifold Z score (max)", "MFE score",
              "G+C Percentage", "Random", "Known sRNA or RINC")

matrixList <- calculateCorrelationMatrix(dat, matNames)
```

```
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties

## Warning in cor.test.default(corInput[, dNames[i] == colnames(corInput)], :
## Cannot compute exact p-value with ties
```

```
rhoMatrix <- matrixList$rhoMatrix
sigMatrix <- matrixList$sigMatrix

heatmap.2(rhoMatrix, cellnote=round(rhoMatrix, digits = 2),
          notecex=1.5,notecol="black", col=rev(redblue(40)),
          density.info="none", trace="none", dendrogram=c("column"),
          symm=F,symkey=T,symbreaks=T, scale="none", key.title = "",
          srtRow=45, adjRow=c(0, 1), srtCol=45, adjCol=c(1,1),
          breaks=(-20:20)/20, margins = c(8, 8), cexRow=1.5,
          cexCol=1.5,font=1)
```



```
if(FALSE){
svglite::svglite(paste0(figure_path, "/svg/heatmap_subset.svg"))
heatmap.2(rhoMatrix, cellnote=sigMatrix,notecex=1.5,notecol="black",
          col=rev(redblue(40)), density.info="none", trace="none",
          dendrogram=c("column"), symm=F,symkey=T,symbreaks=T, scale="none",
          key.title = "", srtRow=45, adjRow=c(0, 1), srtCol=45, adjCol=c(1,1),
          breaks=(-20:20)/20, margins = c(8, 8), cexRow=1.5, cexCol=1.5,font=2)

dev.off()
}
```

UpsetR figure

```
#ML phylogenetic tree built using 16s rRNA for the Gammaproteobacteria
#The full tree was also built for all bacterial genera, but is not displayed
tree <- read.tree(paste0(data_path, 'upsetr.tree'))

  tbl_tree <- as_tibble(tree)

#list of extra Gammaproteobacteria that were
#not included in the RNA-sed analysis
to_drop <- c("Azotobacter", "Marinobacter", "Pseudoalteromonas", "Agarivorans",
             "Vibrio", "Alishewanella", "Aggregatibacter", "Mannheimia",
             "Actinobacillus", "Xenorhabdus", "Providencia", "Proteus",
             "Pantoea", "Brenneria", "Lonsdalea", "Buchnera.", "Wigglesworthia",
             "Sodalis", "Dickeya", "Citrobacter", "Plautiasymbiont",
             "Shewanella", "Moritella", "Moraxella", "Psychrobacter",
             "Methylomonas", "Cycloclasticus", "Methylococcus", "Francisella",
             "Pseudoxanthomonas", "Candidatus", "Plautia", "Methylophaga",
             "Pasteurella", "Salinivibrio")


  sub_tree <- drop.tip(tree, to_drop)

    tbl_sub_tree <- as_tibble(sub_tree)


p <- ggtree(sub_tree) +
  geom_tiplab(align = T) +
  xlim(0, 0.35)

p
if(FALSE){
ggsave(filename = paste0(figure_path, "SVG/upsetr_subset_tree.svg"),
       plot = p, width = 8, height = 16)
}
p <- ggtree(tree) +
  geom_tiplab(align = T) +
  xlim(0, 0.5)

p
if(FALSE){
ggsave(filename = paste0(figure_path, "SVG/upsetr_tree.svg"),
       plot = p, width = 8, height = 16)
}
```

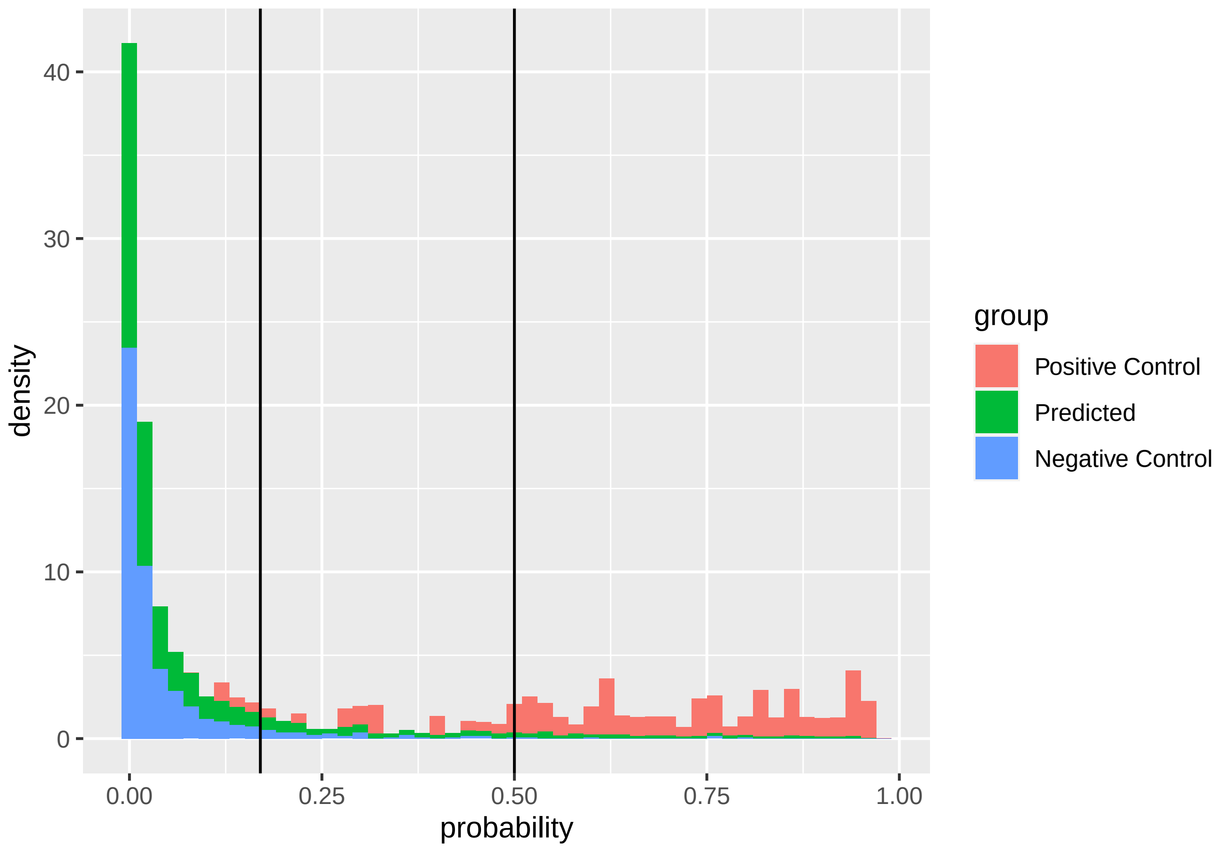
```r
#matrix of bool values indicating for each RNA if it was found in a genus
load(paste0(data_path, "upsetSubsetPC.Rda"))
load(paste0(data_path, "upsetSubsetPredicted.Rda"))

#list of extra Gammaproteobacteria that were
#not included in the RNA-sed analysis
genera_arranged <- c("Lysobacter", "Stenotrophomonas", "Xylella", "Xanthomonas",
                     "Methylomicrobium", "Pseudomonas", "Acinetobacter",
                     "Alteromonas", "Photorhabdus", "Yersinia", "Erwinia",
                     "Edwardsiella", "Serratia", "Klebsiella", "Enterobacter",
                     "Salmonella", "Shigella", "Escherichia")


upset_pred <- select_columns_by_list(upsetSubsetPredicted, genera_arranged)
upset_pc <- select_columns_by_list(upsetSubsetPC, genera_arranged)

UpSetR::upset(upset_pc, sets = colnames(upset_pc),
              mb.ratio = c(0.55, 0.45), order.by = "freq", keep.order = T)
UpSetR::upset(upset_pred, sets = colnames(upset_pred),
              mb.ratio = c(0.55, 0.45), order.by = "freq", keep.order = T)



if(FALSE){
svglite(filename=paste0(figure_path, "SVG/upsetr_pc.svg"))
UpSetR::upset(upset_pc, sets = colnames(upset_pc),
              mb.ratio = c(0.55, 0.45), order.by = "freq", keep.order = T)
dev.off()
svglite(filename=paste0(figure_path, "SVG/upsetr_pred.svg"))
UpSetR::upset(upset_pred, sets = colnames(upset_pred),
              mb.ratio = c(0.55, 0.45), order.by = "freq", keep.order = T)
dev.off()
}

#matrix of bool values indicating for each RNA if it was found in a genus
load(paste0(data_path, "upsetSubsetPC.Rda"))
load(paste0(data_path, "upsetSubsetPredicted.Rda"))


#list of extra Gammaproteobacteria that were not included in the RNA-sed analyis
genera_arranged <- c("Lysobacter", "Stenotrophomonas", "Xylella", "Xanthomonas",
                     "Methylomicrobium", "Pseudomonas", "Acinetobacter",
                     "Alteromonas", "Photorhabdus", "Yersinia", "Erwinia",
                     "Edwardsiella", "Serratia", "Klebsiella", "Enterobacter",
                     "Salmonella", "Shigella", "Escherichia")

upset_pred <- select_columns_by_list(upsetSubsetPredicted, genera_arranged)
upset_pc <- select_columns_by_list(upsetSubsetPC, genera_arranged)

pc_freq <- colSums(upset_pc)
pred_freq <- colSums(upset_pred)

pc_freq <- vector_to_dataframe(pc_freq, 'genera')
```

```r
pred_freq <- vector_to_dataframe(pred_freq, 'genera')

pc_freq <- pc_freq %>% dplyr::rename(freq = vec) %>% mutate(group ='pc')
pred_freq <- pred_freq %>% dplyr::rename(freq = vec) %>% mutate(group ='pred')

freq_data <-pc_freq %>% bind_rows(pred_freq)

freq_data$genera <- factor(freq_data$genera, levels = unique(freq_data$genera))

p <- ggplot() +
  geom_bar(data = freq_data,
           aes(y = genera, x = freq, group = group, fill = group),
           stat = 'identity', position = 'dodge')

p

if(FALSE){
  ggsave(filename = paste0(figure_path, 'SVG/rnas_frequency.svg'),
  plot = p, width = 8, height = 16)
}
```

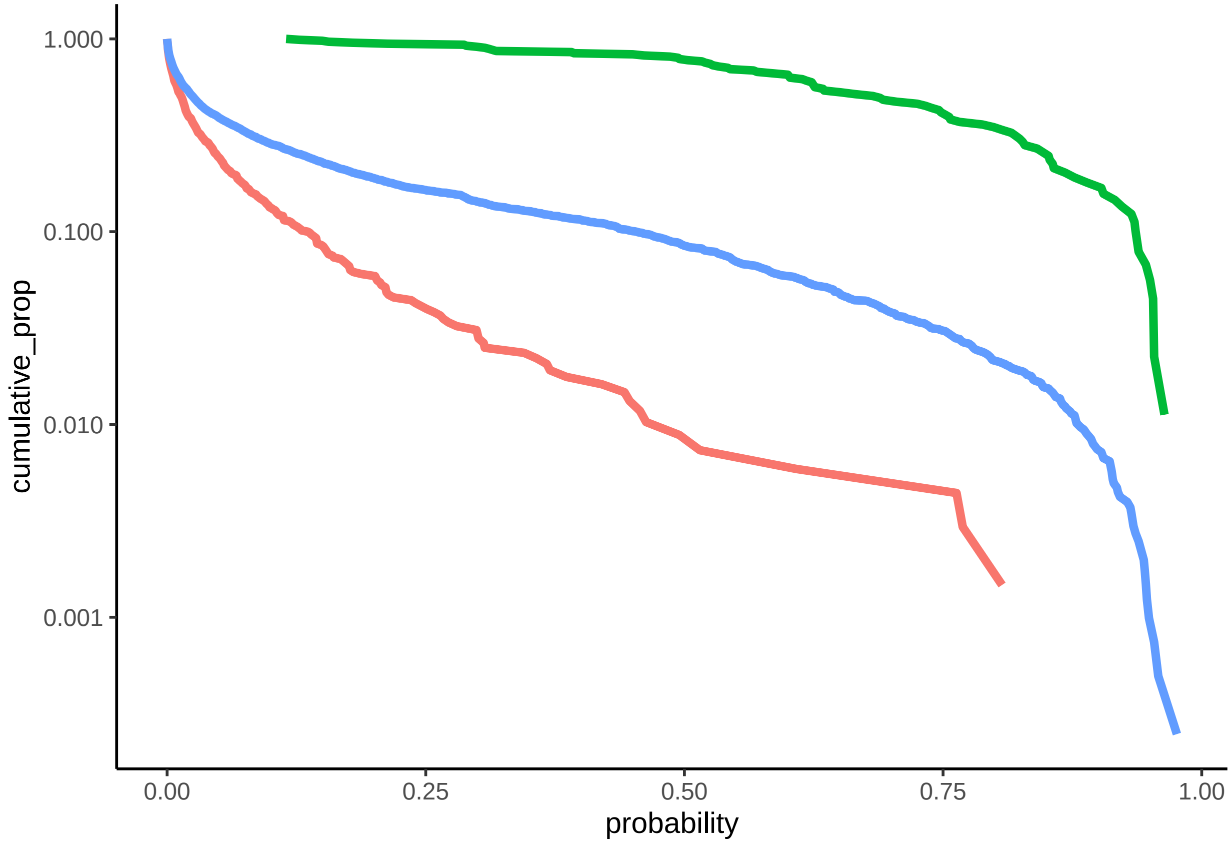
Random forest interpretation figure

- The output of the RF classifier for the test data and the predicted RUFs

```r
load("~/bin/manuscript/data/predDat.Rda")
load("~/bin/PhD/Chapter_4/chapter_4_files/validatation2.Rda")
#select the desired columns from the predicted data and validatation data

probDat <- predDat %>%
  select(probability, ID, group, srna.counts.2) %>%
  bind_rows(validation2 %>% select(probability, ID, group, srna.counts.2))

#not sure if setting factors will break anything so using another data frame
plotDat <- probDat
plotDat$group <- factor(plotDat$group,
                        levels = c('Positive Control',
                                   'Predicted',
                                   'Negative Control'))

#plot histogram of the probabilities
p <- ggplot() +
  geom_histogram(data = plotDat,
                 aes(x = probability,
                     y = ..density..,
                     group = group,
                     fill = group),
                 binwidth = 0.02) +
  geom_vline(xintercept = 0.17) +
  geom_vline(xintercept = 0.5)
p
```



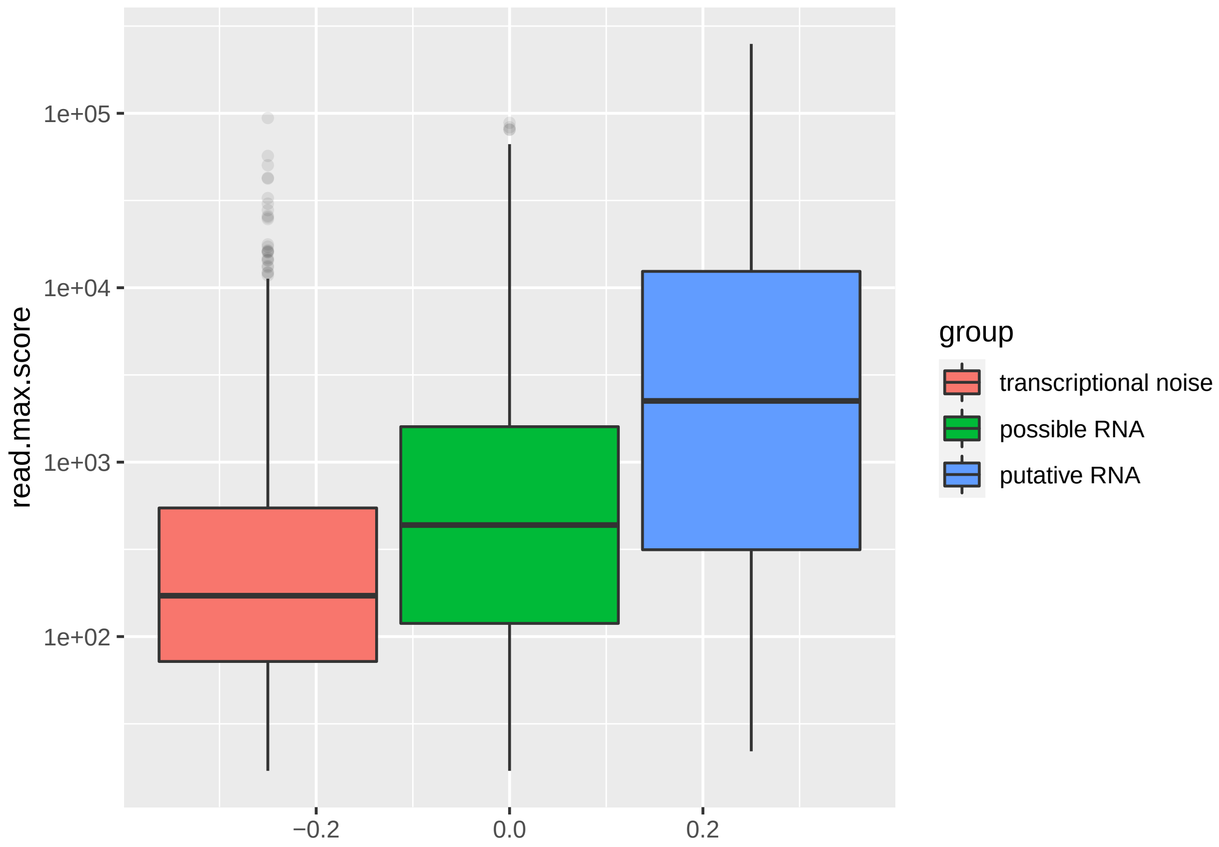
```
if(FALSE){
  ggsave(filename = paste0(figure_path, "SVG/histogram_probabilities.svg"),
         plot = p, width = 178, height = 155, units = "mm")
}

#get the cumulative counts of the number of alignments as probability increases
countsCumul <- cumulativeCounts(dists = probDat,
                                smooth = F,
                                target_column = 'probability')

##produces plot to use for figure showing probability results
p <- ggplot() +
  geom_line(data = countsCumul, aes(x= probability,
                                    y = cumulative_prop,
                                    group = group,
                                    colour = group),
            size = 1.5,
            show.legend = F) +
  scale_y_continuous(trans = 'log10')
p + theme_classic()
```



```
if(FALSE){
  ggsave(filename = paste0(figure_path, "SVG/cumulative_probabilities.svg"),
                          plot = p, width = 178, height = 155, units = "mm")
}
```

- The predicted RUFs were split into transcriptional noise, possible RNA, and putative RNA.
- The categories were then compared to see if there was a difference in the level of transcription between the groups.

```
load("~/bin/manuscript/data/predDat.Rda")

#create groups withing the predDat based on the probability thresholds
predDat <- predDat %>%
  mutate(group = ifelse(probability <0.17,
                        'transcriptional noise',
                        ifelse(probability <= 0.81,
                               'possible RNA',
                               'putative RNA')))
predDat$group = factor(predDat$group, levels = c("transcriptional noise",
                                                 "possible RNA",
                                                 "putative RNA"))
#boxplot of transcription levels for each group
p <- ggplot(data = predDat) +
  geom_boxplot(aes(y = read.max.score, group = group, fill = group),
               outlier.alpha = 0.1) +
  scale_y_continuous(trans = 'log10')
p
```



```
if(FALSE){
  ggsave(filename = paste0(figure_path, "SVG/trascription-comparison.svg"),
         plot = p, width = 450, height = 307, units = "mm")
}
#run wilcox test comparing the transcriptional noise and the other groups
wilcox.test(x = predDat$read.max.score[predDat$group == "transcriptional noise"],
            y = predDat$read.max.score[predDat$group == "putative RNA"])
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  predDat$read.max.score[predDat$group == "transcriptional noise"] and predDat$read.max.score[p
## W = 53040, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(x = predDat$read.max.score[predDat$group == "transcriptional noise"],
            y = predDat$read.max.score[predDat$group == "possible RNA"])
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
```

```
## data:  predDat$read.max.score[predDat$group == "transcriptional noise"] and predDat$read.max.score[p
## W = 861696, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0

#TODO this is not used
load("~/bin/manuscript/data/predDat.Rda")
load("~/bin/PhD/Chapter_4/chapter_4_files/validatation2.Rda")

#select the desired columns from the predicted data and validatation data
probDat <- predDat %>%
  select(probability, ID, group, srna.counts.2) %>%
  bind_rows(validation2 %>%
              select(probability, ID, group, srna.counts.2))

#not sure if setting factors will break anything so using another data frame
plotDat <- probDat
plotDat$group <- factor(plotDat$group,
                        levels = c('Positive Control',
                                   'Predicted',
                                   'Negative Control'))
dat <- allScores(plotDat, 0.01, 'probability')

#get the FNR and PPV values at each threshold
if(FALSE){
  for(i in seq(0,1, by=0.01)){
    scores <- scoreProbabities(plotDat,
                               threshold = i,
                               target_column = 'probability')
    print(paste0(i, ': ', scores$fnr, ', ', scores$ppv))
  }
}


scores <- scoreProbabities(plotDat,
                           threshold = 0.17,
                           target_column = 'probability')
printListSubset(scores,
                vec = c('ppv', 'fnr', 'pred_pos', 'pred_pct'),
                startText = 'p > 0.17', round_val = 3)


scores <- scoreProbabities(plotDat,
                           threshold = 0.5,
                           target_column = 'probability')
printListSubset(scores,
                vec = c('ppv', 'fnr', 'pred_pos', 'pred_pct', 'pc_pct'),
                startText = 'p > 0.5', round_val = 3)


scores <- scoreProbabities(plotDat,
                           threshold = 0.81,
                           target_column = 'probability')
printListSubset(scores,
                vec = c('ppv', 'fnr', 'pred_pos', 'pred_pct', 'pc_pct'),
                startText = 'p > 0.81', round_val = 3)

#TODO this is not used
load("~/bin/manuscript/data/predDat.Rda")
load("~/bin/PhD/Chapter_4/chapter_4_files/validatation2.Rda")

#select the desired columns from the predicted data and validatation data
probDat <- predDat %>%
  select(probability, ID, group, srna.counts.2) %>%
  bind_rows(validation2 %>%
              select(probability, ID, group, srna.counts.2))

#not sure if setting factors will break anything so using another data frame
plotDat <- probDat
plotDat$group <- factor(plotDat$group,
                        levels = c('Positive Control',
                                   'Predicted',
                                   'Negative Control'))
dat <- allScores(plotDat, 0.01, 'probability')

ggplot(data = dat) +
  geom_line(aes(x = threshold, y = ppv), color = 'blue') +
  geom_line(aes(x = threshold, y = fnr), color = 'red')


roc.curve(response = plotDat$group[plotDat$group != 'Predicted'],
          predicted = plotDat$probability[plotDat$group != 'Predicted'])

ggplot(data = dat) +
  geom_line(aes(x = fnr, y = specificity), color = 'blue') +
  geom_line(aes(x = fpr, y = sensitivity), color = 'red')

#TODO this is not used
#is there a difference between low scoring (p < 0.17 and high scoring p > 0.5) predicted rufs?


load("~/bin/manuscript/data/predDat.Rda")
load("~/bin/Phd/Chapter_4/chapter_4_files/validatation2.Rda")
##function written using max_dist as column name so each variable needs to  be renamed to this before us

featuresSelected <- predDat %>%
  mutate(group = ifelse(probability > 0.75, 'high', ifelse(probability <= 0.17, 'low', 'drop'))) %>%
  filter(group != 'drop') %>% select(group, distance, read.max.score,
                                     cov.min.eval, z_max, motif.max.score,
                                     alifold_cov_score, mfe.score,
                                     gc.score) %>%
  bind_rows(validation2 %>% filter(group == 'Positive Control') %>%
              select(group, distance, read.max.score,
                                     cov.min.eval, z_max, motif.max.score,
                                     alifold_cov_score, mfe.score,
                                     gc.score))

dat <- featuresSelected %>% dplyr::rename(max_dist = distance) %>% filter(max_dist <= 0.15)
```

```r
distance.p <- cumulativeDistribution(dat, alternative = "two.sided", show.legend = F)
distance.p <- distance.p +
   labs(y = "Cumulative Proportion", x = "Evolutionary distance")

##known sRNAs and predicted RUFs are only selected if there is read depths. For a fair comparison, RINCs
dat <- featuresSelected %>% select(group, read.max.score) %>% dplyr::rename(max_dist = read.max.score) %
reads.p <- cumulativeDistribution(dat, alternative = 'two.sided', show.legend = F)
reads.p <- reads.p +
   labs(y = "Cumulative Proportion", x = "Total reads")+
  scale_x_continuous(trans = "log10")
#reads.p

dat <- featuresSelected %>% mutate(max_dist = -log(cov.min.eval))
rscape.p <- cumulativeDistribution(dat)
rscape.p <- rscape.p +
   labs(y = "Cumulative Proportion", x = "Rscape covariance score")
# rscape.p

##none of the z scores are greater than 3, so the NA value of 10 is changed to 3 (then the negaive is ta
dat <- featuresSelected %>% mutate(max_dist = ifelse(z_max == 10, -3, -z_max)) %>% select(group, max_dis
z.p <- cumulativeDistribution(dat, show.legend = F)
z.p <- z.p +
   labs(y = "Cumulative Proportion", x = "Alifold z-score (negative energy)")

# z.p
##selected a window where the values are easier to visualise. This has removed 16 known sRNAs and 9 pred
dat <- featuresSelected %>% dplyr::rename(max_dist = motif.max.score) %>% filter(max_dist < 1000)
motif.p <- cumulativeDistribution(dat)
motif.p <- motif.p +
   labs(y = "Cumulative Proportion", x = "Motif score")

dat <- featuresSelected %>% mutate(max_dist = -alifold_cov_score)
alifold.cov.p <- cumulativeDistribution(dat)
alifold.cov.p <- alifold.cov.p +
   labs(y = "Cumulative Proportion", x = "Alifold covariance score")

dat <- featuresSelected %>% mutate(max_dist = -mfe.score)
mfe.p <- cumulativeDistribution(dat, alternative = 'two.sided')
mfe.p <- mfe.p +
   labs(y = "Cumulative Proportion", x = "MFE score (negative energy)")


dat <- featuresSelected %>% arrange(gc.score) %>% mutate(gc.score = round(gc.score))

highCounts <- dat %>% filter(group == "high") %>% group_by(gc.score) %>% summarise(count = n()) %>% arra
lowCounts <- dat %>% filter(group == "low") %>% group_by(gc.score) %>% summarise(count = n()) %>% arrang

highTotal <- dat %>% filter(group == "high") %>% nrow()
lowTotal <- dat %>% filter(group == "low") %>% nrow()

highGC <- zoo::zoo(highCounts$count)
lowGC <- zoo::zoo(lowCounts$count)
```

```r
smoothHigh <- zoo::rollapply(highGC, width = 10, by = 1, FUN = mean, align = "center", partial = T)
smoothLow <- zoo::rollapply(lowGC, width = 10, by = 1, FUN = mean, align = "center", partial = T)

smoothHigh <- as.data.frame(smoothHigh) %>% mutate(x = row_number() -1) %>% mutate(group = "high") %>%
smoothLow <- as.data.frame(smoothLow) %>% mutate(x = row_number() -1) %>% mutate(group = "low") %>% dply

pcCounts <- dat %>% filter(group == "Positive Control") %>% group_by(gc.score) %>% summarise(count = n(
pcTotal <- dat %>% filter(group == "Positive Control") %>% nrow()
pcGC <- zoo::zoo(pcCounts$count)
smoothPC <- zoo::rollapply(pcGC, width = 10, by = 1, FUN = mean, align = "center", partial = T)
smoothPC <- as.data.frame(smoothPC) %>% mutate(x = row_number() -1) %>% mutate(group = "Positive Contro

smoothGC <- smoothHigh %>%  bind_rows(smoothLow, smoothPC)

gc.p <- ggplot() +
  geom_path(data = smoothGC, aes(x = x, y = y, group = group, color = group), size = 1, show.legend = F



all.p <- ggarrange(distance.p, reads.p, mfe.p, z.p, rscape.p, alifold.cov.p,  motif.p, gc.p + rremove("
          labels = LETTERS[1:9],
          ncol = 3, nrow = 3)

all.p


if(FALSE){
ggsave(filename = paste0(figure_path, "SVG/separated_distributions.svg"), plot = all.p, width = 450, he
}
```