

ANEXO

Análisis estadístico de términos y anotaciones biológicas sobre genes humanos para identificación, significación, redundancia, grado de información y asociaciones derivadas (redes)

Realizado por
Cristina Silvia Cornejo Rodríguez

Grado en Estadística

Contenido

1 Preparación de los datos	3
2 Descriptiva	4
BOXPLOT	4
DENSITY PLOT	4
3 Análisis de expresión diferencial	5
4 Análisis de Enriquecimiento Funcional.....	8
1. SEA (pval por cada termino)	8
DAVID:.....	8
GENECODIS.....	8
Scatter plot GO-CC.....	10
2. MEA (pval por cada conjunto términos).....	11
Scatter plot GOCCyBP	11
3. GSEA	12
5 GRAFOS.....	15
1. Grafo Coanotaciones: a partir anotaciones-genes	15
SEA:.....	16
MEA:	18
GSEA:	19
2. Grafo Coexpresión: a partir matriz expresión	21
3. Grafo Aracne.....	23

Para el trabajo se ha utilizado el lenguaje de programación de R (RStudio).

1 Preparación de los datos

Este trabajo parte de unos datos de expresión (20.501 genes y 803 muestras), y los resultados los dos métodos de deconvolución (xCELL y ESTIMATE), estos objetos me dan las clases fenotípicas de las muestras de ImmuneScore (1 : lowIS, 2:highIS). Me quedo con las muestras que coinciden con low y high immune score para los resultados de ambos métodos. Con esto finalmente, obtengo una matriz de expresión de 20.501 genes y 516 muestras con las clases fenotípicas correspondientes a cada muestra.

```
library(rlang)
```

```
library(vctrs)
```

```
library(tidyverse)
```

```
load("C:/Users/merlu/Desktop/TFG/Rdata_ImmuneScore/res_ESTIMATE_immuneScore.Rdata")
```

```
load("C:/Users/merlu/Desktop/TFG/Rdata_ImmuneScore/res_xcellRPKM_immuneScore.Rdata")
```

```
load("C:/Users/merlu/Desktop/TFG/Rdata_ImmuneScore/exprs_BreastRNA.Rdata")
```

```
#LIMPIEZA: matriz de expresion con solo valores de grupos coincidentes
```

```
#LOW Immune-Score: indice muestras que nos quedamos
```

```
#HIGH Immune-Score: indice muestras que nos quedamos
```

```
lowIS <- which(KM_IMMUNEscore_xCell[["patientExpr"]] == 1 &  
KM_IMMUNEscore_ESTIMATE[["patientExpr"]] == 1)
```

```
length(lowIS) #281
```

```
highIS <- which(KM_IMMUNEscore_xCell[["patientExpr"]] == 2 &  
KM_IMMUNEscore_ESTIMATE[["patientExpr"]] == 2)
```

```
length(highIS) #235
```

```
grupos <- KM_IMMUNEscore_xCell[["patientExpr"]][c(lowIS, highIS)] #me quedo con los  
coincidentes
```

```
expresionn <- exprs_803[, c(lowIS, highIS)]
```

```
# View(expresionn)
```

```
table(grupos)
```

```
grupos <- ifelse(grupos == 1, "lowIS", "highIS"); table(grupos)
```

2 Descriptiva

Con la matriz de expresión y los grupos de las muestras según la clase fenotípica se realiza una breve descriptiva visualizando boxplots y gráfico de densidades

```
load("C:/Users/merlu/Desktop/TFG/grupos.RData")
```

```
grupos<-as.factor(grupos)
```

```
load("C:/Users/merlu/Desktop/TFG/expressiomatrix.RData")
```

```
mean(median(expresionn[,1:516]))
```

BOXPLOT

```
#boxplot(expresionn, boxwex = 0.7, notch = T, main = title, las = 2, outline = F, cex.axis = 0.7)
```

```
title <- paste("Diagrama de cajas del experimento")
```

```
#BOXPLOT separado por color grupos
```

```
colores <- palette(c("blue","red","green","orange","grey","purple","yellow"))
```

```
boxplot(expresionn, boxwex = 0.7, notch = T, main = title, las = 2,
```

```
      col = as.factor(grupos), outline = F, cex.axis = 0.7, border=c("NA", "black"))
```

```
legend("top", legend = levels(grupos), xpd = T, inset = -0.055, bg = NULL,
```

```
      fill = colores, cex = 0.6, horiz = T, bty = "n", text.width = 10)
```

DENSITY PLOT

```
plot(density(expresionn[,1]), ylim = c(0, 0.2), xlim = c(-1, 18), xlab = "x", ylab = "Densidad", main="Density plot")
```

```
for (i in 2:516){
```

```
  lines(density(expresionn[,i]))
```

```
}
```

3 Análisis de expresión diferencial

A continuación, se realiza el análisis de expresión diferencial de limma

```
### EdgeR
library(edgeR)
library(limma)

#Matriz de expresion como objeto DGEList
d0 <- DGEList(counts = expresionn, group = grupos)
# d0

#First we calculate normalization factors. calcNormFactors doesn't
#normalize the data, it just calculates normalization factors for use
#downstream.

#realiza el cálculo de factores de normalización para los datos de expresión génica, lo que
#ayuda a ajustar las diferencias sistemáticas entre las muestras y proporciona datos más
#precisos y comparables para el análisis posterior.
d0 <- calcNormFactors(d0)

#Next we will filter low expressed genes. "Low-expressed" is subjective
#and depends on the dataset.
#Determine which genes have sufficiently large counts to be retained in a statistical analysis.
d <- d0[filterByExpr(d0, min.count=1),]

#We save our resistance data into a new variable
group <- as.factor(grupos)
group %>% table()
```

Aquí está incluido el análisis de escalamiento multidimensional añadido en descriptiva

```
#Next we can check Multidimensional scaling (MDS) plot
```

```
#3'
```

```
plotMDS(d, col = as.numeric(group), pch = 16, cex = .65)
```

```
legend("topright", legend = levels(group), col = 1:length(levels(group)), pch = 16, cex = 0.65)
```

```
#Next we will perform the voom transformation and the calculation of
```

```
#variance weights. First, we specify the model to be fitted. We do this
```

```
#before using voom since voom uses variances of the model residuals
```

```
 #(observed - fitted)
```

```
mm <- model.matrix(~0 + group)
```

```
#pheatmap(mm, cluster_rows = TRUE, cluster_cols = TRUE)
```

```
# mm
```

```
#The above specifies a model where each coefficient corresponds to a group mean
```

```
y <- voom(d, mm, plot = T)
```

```
#More details at
```

```
#<https://genomebiology.biomedcentral.com/articles/10.1186/gb-2014-15-2-r29>
```

```
#The next step is fitting linear models in limma
```

```
fit <- lmFit(y, mm)
```

```
head(coef(fit))
```

```
#Specify which groups to compare, in this case we only have two groups:
```

```
contr <- makeContrasts(grouphighIS - grouplowIS, levels = colnames(coef(fit)))
```

```
contr
```

```
#Estimate contrast for each gene
```

```
tmp <- contrasts.fit(fit, contr)
```

```
#Empirical Bayes smoothing of standard errors (shrinks standard errors  
#that are much larger or smaller than those from other genes towards the  
#average standard error) (see  
#<https://www.degruyter.com/doi/10.2202/1544-6115.1027>)  
tmp <- eBayes(tmp)
```

```
#What genes are most differentially expressed?  
top.table <- topTable(tmp, sort.by = "P", n = Inf)  
head(top.table, 20)  
top250table <- head(top.table, 250); top250table
```

```
#How many DE genes are there?
```

```
length(which(top.table$adj.P.Val < 0.01)) #9048  
length(which(top.table$adj.P.Val < 0.000001))  
length(which(top.table$adj.P.Val < 0.0000000000000001))
```

```
#Write top.table to a file
```

```
top.table$Gene <- rownames(top.table)  
top.table <- top.table[,c("Gene", names(top.table)[1:6])]
```

```
write.table(top.table, file = "genes.txt", row.names = F, sep = "\t", quote = F)  
write.table(top.table[1:250,1], file = "nombresgenesDE250.txt", row.names = F, sep = "\t",  
quote = F)
```

4 Análisis de Enriquecimiento Funcional

A continuación se realizan los tres tipos de análisis de enriquecimiento funcional

1. SEA (pval por cada termino)

Se quiere ver las salidas de los análisis de SEA con DAVID y con Genecodis.

DAVID:

Desde la web se puede observar la tabla con total de genes anotados en cada espacio.

```
DAVID.CHART <- read.delim("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment Analysis/davidSEA/DAVID-CHART.txt")
```

```
DAVID.CLUSTER <- read.delim("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment Analysis/davidSEA/DAVID-CLUSTER.txt", header=FALSE, comment.char="#")
```

```
DAVID.TABLE <- read.delim2("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment Analysis/davidSEA/DAVID-TABLE.txt")
```

GENECODIS

Tras la salida de la realización del análisis de SEA con Genecodis, se quiere graficar con diagramas de Venn el número de genes anotados en diferentes espacios de anotación: un diagrama de Venn con espacios de anotación GO-CC, Reactome, Wikipathways, y otro diagrama de Venn con espacio de anotación GO con sus diferentes ontologías.

###diagrama de venn: GOCC,KEGG, REACTOME, INTERPRO ----

#genes incluidos de cada espacio

```
gtableGOBP <- read.delim2("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment Analysis/genecodisSEA/gtableGOBP.txt")
```

```
gtableGOBP<-gtableGOBP$Official.Symbol
```

```
gtableGOCC <- read.delim2("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment Analysis/genecodisSEA/gtableGOCC.txt")
```

```
gtableGOCC <- gtableGOCC$Official.Symbol
```

```
gtableGOMF <- read.delim2("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment
Analysis/genecodisSEA/gtableGOMF.txt")
```

```
gtableGOMF <- gtableGOMF$Official.Symbol
```

```
gtableKEGG <- read.delim2("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment
Analysis/genecodisSEA/gtableKEGG.txt")
```

```
gtableKEGG <- gtableKEGG$Official.Symbol
```

```
gtableREACTOME <- read.delim2("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment
Analysis/genecodisSEA/gtableREACTOME.txt")
```

```
gtableREACTOME <- gtableREACTOME$Official.Symbol
```

```
gtableWIKIPATHW <- read.delim2("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment
Analysis/genecodisSEA/gtableWIKIPATHW.txt")
```

```
gtableWIKIPATHW <- gtableWIKIPATHW$Official.Symbol
```

```
library(VennDiagram)
```

```
# Crear el diagrama de Venn GO's
```

```
venn.diagram(
  x = list(gtableGOBP, gtableGOCC, gtableGOMF),
  category.names = c("GOBP", "GOCC", "GOMF"),
  filename = "venn_genecodisGO.png", # Nombre del archivo de salida (opcional)
  output = "file" # Especifica que el diagrama se guardará en un archivo
)
```

```
# Crear el diagrama de Venn GOCC", "KEGG", "REACTOME", "WIKIPATHWAYS
```

```
venn.diagram(
  x = list(gtableGOCC,gtableKEGG,gtableREACTOME,gtableWIKIPATHW ),
  category.names = c("GOCC", "KEGG", "REACTOME", "WIKIPATHWAYS"),
  filename = "venn_genecodis4.png", # Nombre del archivo de salida (opcional)
  output = "file" # Especifica que el diagrama se guardará en un archivo
)
```

```
####ver pval enriquecimiento ----
```

```
genecodisGO_BP <- read.delim("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment  
Analysis/genecodisSEA/enrich-input1-GO_BP.tsv")
```

```
genecodisGO_CC <- read.delim("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment  
Analysis/genecodisSEA/enrich-input1-GO_CC.tsv")
```

```
genecodisGO_MF <- read.delim("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment  
Analysis/genecodisSEA/enrich-input1-GO_MF.tsv")
```

```
genecodisKEGG <- read.delim("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment  
Analysis/genecodisSEA/enrich-input1-KEGG.tsv")
```

```
genecodisReactome <- read.delim("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment  
Analysis/genecodisSEA/enrich-input1-Reactome.tsv")
```

```
genecodisWikiPathways <- read.delim("C:/Users/merlu/Desktop/TFG/1 Pathways  
Enrichment Analysis/genecodisSEA/enrich-input1-WikiPathways.tsv")
```

```
summary(genecodisGO_CC)
```

```
length(which(genecodisGO_CC$pval_adj<=0.05))
```

Scatter plot GO-CC

Realización de un Scatter plot en el espacio de anotación GO-CC

```
#solo las filas con 1 anotación y mas de 5 gen
```

```
#g2enecodisGO_CC <- genecodisGO_CC[which(str_count(genecodisGO_CC$genes, pattern  
= ",")>4),]
```

```
#solo 90 filas ¿?
```

```
g2enecodisGO_CC <- genecodisGO_CC[1:90,]
```

```
# Create a data frame
```

```
data <- data.frame(annotation_names=g2enecodisGO_CC$annotation_id,  
enrichment_scores= g2enecodisGO_CC$relative_enrichment, gene_counts=  
g2enecodisGO_CC$genes_found, pval_adj=g2enecodisGO_CC$pval_adj)
```

```
# Create the scatter plot
ggplot(data, aes(x = enrichment_scores, y = annotation_names, color = pval_adj)) +
  geom_point(aes(size = gene_counts)) +
  scale_size(range = c(2, 10)) +
  scale_color_gradient(low = "blue", high = "red") +
  labs(x = "Enrichment Scores", y = "Annotation", title = "Enrichment Analysis") +
  theme_minimal()
```

2. MEA (pval por cada conjunto términos)

Realizar el análisis de MEA.

```
meaGOBPCC <- read.delim("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment
Analysis/MEA/enrich-input1-CoAnnotation-GO_BP_GO_CC.tsv")
```

```
length(which(meaGOBPCC$pval_adj<0.05))
```

```
summary(meaGO_CCBP)
```

Scatter plot GOCCyBP

#solo 90 filas ¿?

```
meaGO_CCBP<-meaGOBPCC[1:90,]
```

Create a data frame

```
datamea <- data.frame(annotation_names=meaGO_CCBP$annotation_id,
enrichment_scores= meaGO_CCBP$relative_enrichment,gene_counts=
meaGO_CCBP$genes_found, pval_adj=meaGO_CCBP$pval_adj)
```

Create the scatter plot

```
library(ggplot2)
```

```
ggplot(datamea, aes(x = enrichment_scores, y = annotation_names, color = pval_adj)) +
```

```
geom_point(aes(size = gene_counts)) +
scale_size(range = c(2, 10)) +
scale_color_gradient(low = "blue", high = "red") +
labs(x = "Enrichment Scores", y = "Annotation", title = "Enrichment Analysis") +
theme_minimal()
```

3. GSEA

Realizar el análisis de GSEA

```
## fgsea (https://bioinformatics-core-shared-training.github.io/cruk-summer-school-2018/RNASeq2018/html/06\_Gene\_set\_testing.nb.html) ----
```

```
library(org.Hs.eg.db)
```

```
library(Seurat)
```

```
library(tidyverse)
```

```
library(fgsea)
```

```
genes <- read.delim("C:/Users/merlu/Desktop/TFG/genes.txt")
```

```
fold_changes <- genes$logFC
```

```
names(fold_changes) <- genes$Gene
```

```
#Create ranks basado en logFC
```

```
barplot(sort(fold_changes, decreasing = T), cex.names = 0.5)
```

```
# Load GSEA gene sets: http://www.gsea-msigdb.org/gsea/msigdb/index.jsp
```

```
GObp <- fgsea::gmtPathways("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment Analysis/GSEA/c5.go.bp.v2023.1.Hs.symbols.gmt")
```

```
GOMf <- fgsea::gmtPathways("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment Analysis/GSEA/c5.go.mf.v2023.1.Hs.symbols.gmt")
```

```
GOcc <- fgsea::gmtPathways("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment  
Analysis/GSEA/c5.go.cc.v2023.1.Hs.symbols.gmt")
```

```
GOall <-fgsea::gmtPathways("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment  
Analysis/GSEA/c5.go.v2023.1.Hs.symbols.gmt")
```

```
GO_gene_sets <- c(GObp, GOMf, GOcc)
```

```
Allgenesets <- fgsea::gmtPathways("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment  
Analysis/GSEA/msigdb.v2023.1.Hs.symbols.gmt")
```

```
# GSEA analysis
```

```
gsea_Myo <- fgsea(pathways = GOcc, #solo GOcc?  
  stats = fold_changes,  
  eps = 0.0,  
  minSize=15,  
  maxSize=500)
```

```
save(gsea_Myo , file = "gsea_Myo .RData")
```

```
head(gsea_Myo[order(pval), ])
```

```
# make a table plot for a bunch of selected pathways:
```

```
topPathwaysUp <- gsea_Myo[ES > 0][head(order(pval), n=10), pathway]  
topPathwaysDown <- gsea_Myo[ES < 0][head(order(pval), n=10), pathway]  
topPathways <- c(topPathwaysUp, rev(topPathwaysDown))  
plotGseaTable(GOcc [topPathways], fold_changes, gsea_Myo,  
  gseaParam=0.5)
```

```
# make an enrichment plot for a pathway (el mas enriquecido por ejmp)
```

```
plotEnrichment(GOcc[["GOCC_EXTERNAL_SIDE_OF_PLASMA_MEMBRANE"]],  
  fold_changes) +  
labs(title="GOCC_EXTERNAL_SIDE_OF_PLASMA_MEMBRANE")
```

```

#PARA TODA ONTOLOGIA GO
{

# GSEA analysis
gsea_Myoall <- fgsea(pathways = GOall, #PARA TODA ONTOLOGIA GO
                    stats = fold_changes,
                    eps = 0.0,
                    minSize=15,
                    maxSize=500)

head(gsea_Myo[order(pval), ])

# make a table plot for a bunch of selected pathways:
topPathwaysUp <- gsea_Myoall[ES > 0][head(order(pval), n=10), pathway]
topPathwaysDown <- gsea_Myoall[ES < 0][head(order(pval), n=10), pathway]
topPathways <- c(topPathwaysUp, rev(topPathwaysDown))
plotGseaTable(GOall [topPathways], fold_changes, gsea_Myoall,
              gseaParam=0.5)

# make an enrichment plot for a pathway (el mas enriquecido por ejmp)
plotEnrichment(GOall[["GOBP_ADAPTIVE_IMMUNE_RESPONSE"]],
              fold_changes) +
labs(title="GOCC_EXTERNAL_SIDE_OF_PLASMA_MEMBRANE")

}

```

5 GRAFOS

1. Grafo Coanotaciones: a partir anotaciones-genes

Creación de funciones para calcular la matriz de indicencias a partir de la salida de genecodis, y otra función para crear matriz de indicencias a partir del output de la librería de R:: fgsea.

```
#funcion creacion matriz de indicencias (0,1)

nombresgenesDE250 <- read.csv("C:/Users/merlu/Desktop/TFG/nombresgenesDE250.txt",
sep="")
```

```
incidenciasgenecodis<-function(dfEA, nombresgenes){
```

```
  df<-matrix(NA, nrow=nrow(dfEA), ncol=length(nombresgenes)) #df filas terminos,
  columnas genes
```

```
  rownames(df)<-dfEA$annotation_id
```

```
  colnames(df)<-nombresgenes
```

```
  library(stringr)
```

```
  #raw data table: df 0,1
```

```
  for (i in 1:nrow(df)){
```

```
    for (j in 1:ncol(df)){
```

```
      if (str_detect(dfEA$genes[i], pattern = colnames(df)[j]) == TRUE){df[i,j]<-1} else
      {df[i,j]<-0}
```

```
    }
```

```
  }
```

```
  return(df)
```

```
}
```

```
incidenciasfgsea<-function(dfEA, nombresgenes){
```

```

df<-matrix(NA, nrow=nrow(dfEA), ncol=length(nombresgenes)) #df filas terminos,
columnas genes

rownames(df)<-dfEA$pathway
colnames(df)<-nombresgenes


library(stringr)
#raw data table: df 0,1
for (i in 1:nrow(df)){
  for (j in 1:ncol(df)){
    if (str_detect(dfEA$leadingEdge[i], pattern = colnames(df)[j]) == TRUE){df[i,j]<-1} else
{df[i,j]<-0}
  }
}
return(df)
}

```

SEA:

Código para calcular el grafo de coanotación a partir del resultado del SEA y cálculo de comunidades.

```

genecodisGO_CC <- read.delim("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment
Analysis/genecodisSEA/enrich-input1-GO_CC.tsv")

```

```

View(genecodisGO_CC) #por ejemplo CC

```

#hacerlo con 30 genes?

```

SEAincidencias<-incidenciasgenocodis(dfEA=genecodisGO_CC, nombresgenes =
nombresgenesDE250$x[1:80])

```

#<https://davetang.org/muse/2017/03/16/matrix-to-adjacency-list-in-r/>

```

{

```

```

#matriz similaridad (no correlacion, uso distancias mejor)

```

```

library(lsa)

```

```

my_sim_matrix<-cosine(SEAincidencias)

```

```

# replace upper triangle of the matrix with number that can be used for filtering
my_sim_matrix[upper.tri(my_sim_matrix)] <- 42

library(reshape2)
my_sim_df <- melt(my_sim_matrix)

library(dplyr)
# __Adjacency matrix (weight: similarities :) ---
# filter out the upper matrix values, filter out the self correlations, y los que no hay relacion
my_adj_list <- my_sim_df[my_sim_df$value!=42 & my_sim_df$Var1 != my_sim_df$Var2
& my_sim_df$value!=0,]
names(my_adj_list) <- c('from', 'to', 'weight')
my_adj_list <- my_adj_list[complete.cases(my_adj_list),] #quitar NAs
#View(my_adj_list)
library(igraph)
# create igraph S3 object
net <- graph.data.frame(my_adj_list, directed = FALSE) ##### GRAFO NO
DIRIGIDO
plot(net, layout = layout_components(net), edge.width = E(net)$weight)

# __Newman-Girvan algorithm ---
#The igraph package can detect communities or subgraphs using the Newman-Girvan
algorithm.
# community detection based on edge betweenness (Newman-Girvan)
ceb <- cluster_edge_betweenness(net)
class(ceb)
membership(ceb)# community membership for each node
}
plot(ceb, net) #A graph with each node grouped by community membership.
dendPlot(ceb, mode="hclust") #A simple dendrogram created using hierarchical clustering.

cluster1 <-ceb

```

MEA:

Código para calcular el grafo de coanotación a partir del resultado del MEA y cálculo de comunidades.

```
meaGOBPyCC <- read.delim("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment Analysis/MEA/enrich-input1-CoAnnotation-GO_BP_GO_CC.tsv")
```

```
View(meaGOBPyCC)
```

```
#hacerlo con 30 genes?
```

```
MEAincidencias<-incidenciasgenocodis(dfEA=meaGOBPyCC, nombresgenes = nombresgenesDE250$x[1:80])
```

```
#https://davetang.org/muse/2017/03/16/matrix-to-adjacency-list-in-r/
```

```
{
```

```
#matriz similaridad (no correlacion, uso distancias mejor)
```

```
library(lsa)
```

```
my_sim_matrix<-cosine(MEAincidencias)
```

```
# replace upper triangle of the matrix with number that can be used for filtering
```

```
my_sim_matrix[upper.tri(my_sim_matrix)] <- 42
```

```
library(reshape2)
```

```
my_sim_df <- melt(my_sim_matrix)
```

```
library(dplyr)
```

```
# __Adjacency matrix (weight: similaridades :)) ---
```

```
# filter out the upper matrix values, filter out the self correlations, y los que no hay relacion
```

```
my_adj_list <- my_sim_df[my_sim_df$value!=42 & my_sim_df$Var1 != my_sim_df$Var2 & my_sim_df$value!=0 ,]
```

```
names(my_adj_list) <- c('from', 'to', 'weight')
```

```
my_adj_list <- my_adj_list[complete.cases(my_adj_list),] #quitar NAs
```

```
#View(my_adj_list)
```

```
library(igraph)
```

```
# create igraph S3 object
```

```

net <- graph.data.frame(my_adj_list, directed = FALSE) ##### IMPORTANTE
#plot(net, layout = layout_components(net), edge.width = E(net)$weight)

# __Newman-Girvan algorithm ---
#The igraph package can detect communities or subgraphs using the Newman-Girvan
algorithm.
# community detection based on edge betweenness (Newman-Girvan)
ceb <- cluster_edge_betweenness(net)
class(ceb)
membership(ceb)# community membership for each node
}
plot(ceb, net) #A graph with each node grouped by community membership.
dendPlot(ceb, mode="hclust") #A simple dendrogram created using hierarchical clustering.

cluster2 <-ceb

```

GSEA:

Código para calcular el grafo de coanotación a partir del resultado del GSEA y cálculo de comunidades.

```

load("C:/Users/merlu/Desktop/TFG/1 Pathways Enrichment Analysis/GSEA/gsea_Myo
.RData")

```

```

View(gsea_Myo) #GOcc

```

#hacerlo con 30 genes?

```

GSEAINCIDENCIAS<-incidenciasfgsea(dfEA=gsea_Myo, nombresgenes =
nombresgenesDE250$x[1:80])

```

#<https://davetang.org/muse/2017/03/16/matrix-to-adjacency-list-in-r/>

```

{

```

```

#matriz similaridad (no correlacion, uso distancias mejor)

library(lsa)

my_sim_matrix<-cosine(GSEAincidencias)

# replace upper triangle of the matrix with number that can be used for filtering
my_sim_matrix[upper.tri(my_sim_matrix)] <- 42


library(reshape2)

my_sim_df <- melt(my_sim_matrix)


library(dplyr)

# __Adjacency matrix (weigth: similaridades :)) ---

# filter out the upper matrix values, filter out the self correlations, y los que no hay relacion
my_adj_list <- my_sim_df[my_sim_df$value!=42 & my_sim_df$Var1 != my_sim_df$Var2
& my_sim_df$value!=0 ,]

names(my_adj_list) <- c('from', 'to', 'weight')

my_adj_list <- my_adj_list[complete.cases(my_adj_list),] #quitar NAs

#View(my_adj_list)

library(igraph)

# create igraph S3 object

net <- graph.data.frame(my_adj_list, directed = FALSE) ##### IMPORTANTE

#plot(net, layout = layout_components(net), edge.width = E(net)$weight)


# __Newman-Girvan algorithm ---

#The igraph package can detect communities or subgraphs using the Newman-Girvan
algorithm.

# community detection based on edge betweenness (Newman-Girvan)

ceb <- cluster_edge_betweenness(net)

class(ceb)

membership(ceb)# community membership for each node
}

plot(ceb, net) #A graph with each node grouped by community membership.

```

```
dendPlot(ceb, mode="hclust") #A simple dendrogram created using hierarchical clustering.
```

```
cluster3 <-ceb
```

2. Grafo Coexpresión: a partir matriz expresión

Código para calcular el grafo de coexpresión y cálculo de comunidades.

```
#matriz de coexpresión utilizando la función cor para calcular las correlaciones entre los genes en la matriz de expresión
```

```
#cor_matrix <- cor(t(expresionn)) #20501x20501 genes
```

```
cor_matrix250<-
```

```
cor(t(expresionn[which(rownames(expresionn)%in%nombresgenesDE250$x),]))
```

```
cor_matrix80<-
```

```
cor(t(expresionn[which(rownames(expresionn)%in%nombresgenesDE250$x[1:80]),]))
```

```
umbral <- 0.9
```

```
red_coexpresion <- cor_matrix80 >= umbral #logical
```

```
library(igraph)
```

```
red_grafo <- graph.adjacency(red_coexpresion, mode = "undirected")
```

```
plot(red_grafo)
```

```
# __Newman-Girvan algorithm ---
```

```
#The igraph package can detect communities or subgraphs using the Newman-Girvan algorithm.
```

```
# community detection based on edge betweenness (Newman-Girvan)
```

```
ceb <- cluster_edge_betweenness(red_grafo)
```

```
class(ceb)
```

```
membership(ceb)# community membership for each node
```

```
plot(ceb, red_grafo) #A graph with each node grouped by community membership.  
dendPlot(ceb, mode="hclust") #A simple dendrogram created using hierarchical clustering.
```

```
cluster4 <-ceb
```

Para calcular a mano las intersecciones de genes contenidos en las comunidades creadas de los diferentes clústeres.

```
#intersiones clusters
```

```
cluster1
```

```
table(cluster1$membership)
```

```
cluster1[1][[1]] #1
```

```
cluster1[2][[1]] #2
```

```
cluster2
```

```
table(cluster2$membership)
```

```
cluster2[3][[1]] #3
```

```
cluster2[4][[1]] #4
```

```
cluster3
```

```
table(cluster3$membership)
```

```
cluster3[1][[1]] #5
```

```
cluster3[2][[1]] #6
```

```
cluster3[6][[1]] #7
```

```
cluster4
```

```
table(cluster4$membership)
```

```
cluster4[5][[1]] #8
```

```
cluster4[1][[1]] #9
```

```
cluster4[7][[1]] #10
```

```
length(intersect(cluster4[5][[1]],cluster3[6][[1]] ))
```

```
length(intersect(cluster4[1][[1]],cluster3[6][[1]] ))
```

```
length(intersect(cluster4[7][[1]],cluster3[6][[1]] ))
```

3. Grafo Aracne

El output de este algoritmo .txt, luego es necesario introducirlo a Cytoscape para poder crear la imagen del grafo.

```
#transcriptional regulatory network reverse-engineering with ARACNe3.
```

```
# load the packages
```

```
library(tidyverse)
```

```
library(Rcpp)
```

```
library(BH)
```

```
library(DESeq2)
```

```
library(mblm)
```

```
library(viper)
```

```
library(pROC)
```

```
library(devtools)
```

```
library(DescTools)
```

```
library(NaRnEA)
```

```
sourceCpp(paste("NaRnEA", "data", "ARACNe3", "ljb_apmi_estimator.cpp",  
               sep = "/"))
```

```
figure.output.value <- 1
```

```
sim.seed <- 1
```

```
sig.figs <- 4
```

```
set.seed(sim.seed)
```

```
# set the current putative transcriptional regulators as the Entrez
```

```

# ID transcription factors (TFs) and Entrez ID co-transcription
# factors (coTFs)
cur.regulator.names <- c(readRDS(paste("NaRnEA", "data", "ARACNe3", "Entrez_TF.rds",
                                     sep = "/")), readRDS(paste("NaRnEA", "data", "ARACNe3",
                                     "Entrez_coTF.rds",
                                     sep = "/")))
cur.regulator.names <- sort(unique(cur.regulator.names))

# create the current output directory
cur.output.dir <- "ARACNEEE_Output"
system(paste("mkdir ", cur.output.dir, sep = ""))

##yo: pasar a entrez id o g_111_ ----
#####

library(biomaRt)

ensembl <- useMart("ensembl", dataset = "hsapiens_gene_ensembl") # Crea una conexi3n a
BioMart usando la base de datos de Ensembl

symbol_list <- rownames(expresionn) # Define la lista de s3mbolos de genes
# Obt3n la conversi3n de s3mbolos a Entrez ID
conversion <- getBM(attributes = c("entrezgene_id", "external_gene_name"),
                    filters = "external_gene_name",
                    values = symbol_list,
                    mart = ensembl)

#ordenar, que expresion y conversion se ordenen alfabeticamente
mat.expresion <- expresionn[order(rownames(expresionn)), ] #ordeno
conversion2 <- conversion[order(conversion$external_gene_name),] #ordeno
conversion2 <- conversion2[complete.cases(conversion2),] #(quitar filas NAs)

```

```

#añadir g_111_
nombres <- paste("g_", conversion2$entrezgene_id, "_", sep = "")
# Agregar la columna adicional a la matriz
conversion2$g_numero_ <- nombres

mat.expresion<-mat.expresion[conversion2$external_gene_name,] #me quedo con los genes
que tiene conversion2

#poner rowname g_111_
rownames(mat.expresion)<- conversion2$g_numero_
#finalmente de los 20mil genes oficial, solo renombra 17mil
#####

cur.tcga.entrez.counts.mat <- mat.expresion #matriz expresion de entrez id limpia

# Separate Gene Expression Profiles for Network Reverse-Engineering and Gene Set Analysis
----

# separate gene expression profiles which come from primary tumor
# samples and gene expression profiles which come from adjacent
# normal tissue samples

cur.tcga.tumor.counts.mat <- cur.tcga.entrez.counts.mat[,
which(substr(colnames(cur.tcga.entrez.counts.mat),
start = 13, stop = 15) == "-01")]

cur.tcga.tissue.counts.mat <- cur.tcga.entrez.counts.mat[,
which(substr(colnames(cur.tcga.entrez.counts.mat),

```

```
start = 13, stop = 15) == "-11"]]
```

```
# identify the gene expression profiles which are paired (i.e. which  
# come from a single patient) and separate these from the unpaired  
# gene expression profiles
```

```
paired.barcode.values <- intersect(substr(colnames(cur.tcga.tumor.counts.mat),  
                                     1, 12), substr(colnames(cur.tcga.tissue.counts.mat), 1, 12))
```

```
cur.paired.tumor.counts.mat <- cur.tcga.tumor.counts.mat[, match(paired.barcode.values,  
                                                                substr(colnames(cur.tcga.tumor.counts.mat), 1, 12))]
```

```
saveRDS(cur.paired.tumor.counts.mat, file = paste(cur.output.dir,  
"/TCGA_LUAD_Paired_Tumor_Counts_Mat.rds",  
        sep = ""))
```

```
cur.paired.tissue.counts.mat <- cur.tcga.tissue.counts.mat[, match(paired.barcode.values,  
                                                                substr(colnames(cur.tcga.tissue.counts.mat), 1, 12))]
```

```
saveRDS(object = cur.paired.tissue.counts.mat, file = paste(cur.output.dir,  
"/TCGA_LUAD_Paired_Tissue_Counts_Mat.rds", sep =  
""))
```

```
# set the unpaired primary tumor gene expression profiles to be used
```

```
# for network reverse-engineering with ARACNe3
```

```
cur.network.tumor.counts.mat <- cur.tcga.tumor.counts.mat[,  
match(setdiff(colnames(cur.tcga.tumor.counts.mat),  
              colnames(cur.paired.tumor.counts.mat)),  
      colnames(cur.tcga.tumor.counts.mat))]
```

```
saveRDS(object = cur.network.tumor.counts.mat, file = paste(cur.output.dir,  
"/TCGA_LUAD_Network_Tumor_Counts_Mat.rds",  
        sep = ""))
```

```
#Reverse-Engineer the Context Specific Transcriptional Regulatory Network with ARACNe3
```

```
----
```

```
# normalize network tumor gene expression profiles for sequencing
```

```
# depth (counts per million)
```

```
cur.network.tumor.cpm.mat <- apply(cur.network.tumor.counts.mat, 2, function(x) {
```

```
  y <- 1e+06 * x/sum(x)
```

```
  return(y)
```

```
})
```

```
# copula-transform the network tumor counts per million to simplify
```

```
# downstream calculations
```

```
set.seed(sim.seed)
```

```
cur.network.tumor.copula.mat <- t(apply(cur.network.tumor.cpm.mat, 1, function(x) {
```

```
  y <- rank(x, ties.method = "random")/(1 + length(x))
```

```
  return(y)
```

```
}})
```

```
# subset the regulators to those that are present in the gene expression profiles
```

```
## sub.regulator.names ----
```

```
sub.regulator.names <- cur.regulator.names[which(cur.regulator.names %in%  
                                                  rownames(cur.network.tumor.copula.mat))]
```

```
# create the subnetwork downsampling fold list
```

```
set.seed(sim.seed)
```

```
cur.max.subnetwork.num <- 100
```

```
cur.downsample.proportion <- (1 - exp(-1))
```

```

cur.downsample.num <- ceiling(ncol(cur.network.tumor.copula.mat) *
cur.downsample.proportion)

cur.downsample.fold.list <- lapply(seq(from = 1, to = cur.max.subnetwork.num,
                                     by = 1), function(i) {
  y <- sample(1:ncol(cur.network.tumor.copula.mat), size =
cur.downsample.num,
                                     replace = FALSE)
  return(y)
})

saveRDS(object = cur.downsample.fold.list, file = paste(cur.output.dir,
                                                         "/TCGA_LUAD_", ncol(cur.network.tumor.copula.mat),
"_ARACNe3_Fold_List.rds",
                                                         sep = ""))

```

```

# select the copula-transformed network gene expression profiles for
# the first subnetwork
sub.gep.mat <- cur.network.tumor.copula.mat[, cur.downsample.fold.list[[1]]]

# select enough null regulators to estimate null mutual information
# values for the mutual information null distribution
set.seed(sim.seed)
cur.null.mi.values <- 1e+06
null.regulator.names <- sample(setdiff(rownames(sub.gep.mat), sub.regulator.names),
                                size = ceiling((cur.null.mi.values/nrow(sub.gep.mat))))

# write the copula-transformed gene expression data for the current

```

```

# ARACNe3 subnetwork to the output directory as a properly formatted
# TXT file
sub.gep.output.data <- as.data.frame(cbind(gene = rownames(sub.gep.mat),
                                           sub.gep.mat))
colnames(sub.gep.output.data) <- c("gene", paste("Sample", 1:ncol(sub.gep.mat),
                                                sep = ""))
rownames(sub.gep.output.data) <- NULL
write.table(sub.gep.output.data, file = paste(cur.output.dir, "/exp_mat.txt",
                                              sep = ""), quote = FALSE, sep = "\t", row.names = FALSE,
col.names = TRUE)

# write the subset regulator names to the output directory
write.table(sub.regulator.names, file = paste(cur.output.dir, "/sub_regulator_data.txt",
                                              sep = ""), quote = FALSE, sep = "\t", row.names = FALSE,
col.names = FALSE)

# create the null subsampled and copula-transformed gene expression
# matrix and write to the output directory
set.seed(sim.seed)
null.gep.mat <- t(apply(sub.gep.mat, 1, function(x) {
  y <- sample(1:length(x), size = length(x), replace = FALSE)/(1 + length(x))
  return(y)
}))
dimnames(null.gep.mat) <- dimnames(sub.gep.mat)

null.gep.output.data <- as.data.frame(cbind(gene = rownames(null.gep.mat),
                                           null.gep.mat))
colnames(null.gep.output.data) <- c("gene", paste("Sample", 1:ncol(null.gep.mat),
                                                sep = ""))
rownames(null.gep.output.data) <- NULL
write.table(null.gep.output.data, file = paste(cur.output.dir, "/null_exp_mat.txt",
                                              sep = ""), quote = FALSE, sep = "\t", row.names = FALSE,
col.names = TRUE)

```

```

# write the null regulator names to the output directory
write.table(null.regulator.names, file = paste(cur.output.dir, "/null_regulator_data.txt",
                                              sep = ""), quote = FALSE, sep = "\t", row.names = FALSE,
                                              col.names = FALSE)

# create a dummy mutual information threshold file written to zero
# (necessary for the ARACNe jar to run)
write.table(x = "0", file = paste(cur.output.dir, "/miThreshold_p1E-8_samples",
                                  ncol(sub.gep.mat), ".txt", sep = ""), quote = FALSE, sep = "\t",
                                  row.names = FALSE,
                                  col.names = FALSE)

# compute the null mutual information values with the null gene
# expression profiles and the null regulators use the ARACNe jar
###aracne3.jar.path <- paste("NaRnEA", "data", "ARACNe3", "aracne.jar", sep = "/") ----
aracne3.jar.path <- paste("C:", "Users", "user", "Downloads", "pipeline", "NaRnEA", "data",
                        "ARACNe3", "aracne.jar", sep = "/")

cur.command <- paste("java -Xmx5G -jar ", aracne3.jar.path, " -e ", paste(cur.output.dir,
                                                                    "/null_exp_mat.txt", sep = ""), " -o ",
cur.output.dir, " --tfs ",
                  paste(cur.output.dir, "/null_regulator_data.txt", sep = ""), " --pvalue 1E-8",
                  "--threads 1", " --seed 1", " --nobootstrap", " --nodpi", sep = "")

system(cur.command)

# load in the null mutual information values
cur.null.mi.values <- read.table(file = paste(cur.output.dir, "nobootstrap_network.txt",
                                              sep = "/"), header = TRUE, sep = "\t")
cur.null.mi.values <- as.numeric(cur.null.mi.values$MI)

```

```

# fit a piecewise null model for mutual information using an eCDF and
# robust linear regression regression
set.seed(sim.seed)

tmp.null.y.values <- seq(from = -9, to = -3, length.out = 1000)

tmp.null.x.values <- -1 * quantile(x = (-1 * cur.null.mi.values), probs =
exp(tmp.null.y.values),
                                names = FALSE)

tmp.null.data <- data.frame(x.values = tmp.null.x.values, y.values = tmp.null.y.values)

cur.null.tsr.model <- mblm(y.values ~ x.values, tmp.null.data)


# remove the null mutual information estimation file and estimate the
# mutual information values between the true regulators and all
# potential target genes
cur.command <- paste("rm ", cur.output.dir, "/nobootstrap_network.txt",
                    sep = "")
system(cur.command)

cur.command <- paste("java -Xmx5G -jar ", aracne3.jar.path, " -e ", paste(cur.output.dir,
                                                                    "/exp_mat.txt", sep = ""), " -o ", cur.output.dir, "
--tfs ", paste(cur.output.dir,
                                                                    "/sub_regulator_data.txt", sep = ""), " --pvalue 1E-8", "--threads 1",

```

```

"--seed 1", "--nobootstrap", "--nodpi", sep = "")

system(cur.command)

# load in the true mutual information values
cur.true.mi.values <- read.table(file = paste(cur.output.dir, "nobootstrap_network.txt",
                                             sep = "/"), header = TRUE, sep = "\t")
cur.true.mi.values <- as.numeric(cur.true.mi.values$MI)

# calculate the statistical significance (i.e. right-tailed p-values)
# for the true mutual information values using the eCDF. If the
# right-tailed p-value calculated with the eCDF is less than 0.05,
# recalculate the right-tailed p-value with the robust regression
# null model
cur.true.tsr.log.p.values <- (as.numeric(cur.null.tsr.model$coefficients[2]) *
                             (cur.true.mi.values) + as.numeric(cur.null.tsr.model$coefficients[1]))
cur.true.log.p.values <- log(ecdf(-1 * cur.null.mi.values)(-1 * cur.true.mi.values))
cur.true.log.p.values[which(exp(cur.true.log.p.values) < 0.05)] <-
cur.true.tsr.log.p.values[which(exp(cur.true.log.p.values) <
                                0.05)]

# set an FDR threshold
cur.mi.threshold.fdr.value <- 0.05

# identify the smallest mutual information value which satisfies the
# FDR threshold according to the methodology of Benjamini and
# Hochberg
set.seed(sim.seed)

cur.true.rank.values <- rank(x = cur.true.log.p.values, ties.method = "random")
cur.true.check.values <- ((cur.true.log.p.values + log(length(cur.true.log.p.values)) -
                          log(cur.true.rank.values)) <= log(cur.mi.threshold.fdr.value))

```

```

cur.mi.threshold.value <- min(cur.true.mi.values[which(cur.true.check.values)])

# write the newly computed mutual information threshold to the
# appropriate file in the output directory
write.table(x = cur.mi.threshold.value, file = paste(cur.output.dir, "/miThreshold_p1E-
8_samples",
                                                    ncol(sub.gep.mat), ".txt", sep = ""), quote = FALSE, sep =
"\t", row.names = FALSE,
          col.names = FALSE)

# remove the previous mutual information values from the output
# directory and reverse-engineer the first ARACNe3 subnetwork with
# both network pruning steps
cur.command <- paste("rm ", cur.output.dir, "/nobootstrap_network.txt",
                    sep = "")
system(cur.command)

cur.command <- paste("java -Xmx5G -jar ", aracne3.jar.path, " -e ", paste(cur.output.dir,
                                                                    "/exp_mat.txt", sep = ""), " -o ", cur.output.dir, "
--tfs ", paste(cur.output.dir,
                                                                    "/sub_regulator_data.txt", sep = ""), " --pvalue 1E-8", "--threads 1",
                                                                    " --seed 1", " --nobootstrap", sep = "")

system(cur.command)

# load in the edges for the first ARACNe3 subnetwork
cur.subnetwork.data <- read.table(file = paste(cur.output.dir, "nobootstrap_network.txt",

```

```

        sep = "/"), header = TRUE, sep = "\t")

# extract the edge names and write them to the output directory
cur.subnetwork.edge.values <- paste(as.character(cur.subnetwork.data$Regulator),
                                   as.character(cur.subnetwork.data$Target), sep = "---")

saveRDS(cur.subnetwork.edge.values, file = paste(cur.output.dir,
"aracne3_subnetwork_1_edges.rds",
        sep = "/"))

# begin compiling count data for the ARACNe3 subnetwork edges
sub.loop.edge.values <- cur.subnetwork.edge.values

compiled.edge.counts <- rep(1, times = length(sub.loop.edge.values))
names(compiled.edge.counts) <- sub.loop.edge.values

# check if all regulators have at least 50 targets
cur.regulator.count.values <- sapply(strsplit(x = names(compiled.edge.counts),
        split = "---", fixed = TRUE), function(x) {
        return(x[1])
    })

cur.regulator.count.values <- factor(cur.regulator.count.values, levels = sub.regulator.names)
cur.regulator.count.values <- table(cur.regulator.count.values)

cur.stop.check.value <- (mean(as.numeric(cur.regulator.count.values) >=
        50) == 1)

# continue reverse-engineering ARACNe3 subnetworks until all the
# stopping criterion is met or until all subsampling folds have been
# used
cur.subnetwork.fold.num <- 1

```

```

while (((cur.stop.check.value == 0) & (cur.subnetwork.fold.num <
cur.max.subnetwork.num))) {

  # iterate the current subnetwork fold number
  cur.subnetwork.fold.num <- (cur.subnetwork.fold.num + 1)

  # subset the copula-transformed network gene expression data to
  # the samples for the current subnetwork
  sub.gep.mat <- cur.network.tumor.copula.mat[,
cur.downsample.fold.list[[cur.subnetwork.fold.num]]]

  # write the subsampled and copula-transformed gene expression
  # data to the output directory as a properly formatted TXT file
  # after removing the previous gene expression profile matrix
  cur.command <- paste("rm ", cur.output.dir, "/exp_mat.txt", sep = "")
  system(cur.command)

  sub.gep.output.data <- as.data.frame(cbind(gene = rownames(sub.gep.mat),
                                             sub.gep.mat))
  colnames(sub.gep.output.data) <- c("gene", paste("Sample", 1:ncol(sub.gep.mat),
                                                    sep = ""))
  rownames(sub.gep.output.data) <- NULL
  write.table(sub.gep.output.data, file = paste(cur.output.dir, "/exp_mat.txt",
                                                sep = ""), quote = FALSE, sep = "\t", row.names = FALSE,
col.names = TRUE)

  # reverse-engineer the current ARACNe3 subnetwork
  cur.command <- paste("rm ", cur.output.dir, "/nobootstrap_network.txt",
                      sep = "")
  system(cur.command)

  cur.command <- paste("java -Xmx5G -jar ", aracne3.jar.path, " -e ",
                      paste(cur.output.dir, "/exp_mat.txt", sep = ""), " -o ", cur.output.dir,

```

```
" --tfs ", paste(cur.output.dir, "/sub_regulator_data.txt", sep = ""),  
" --pvalue 1E-8", "--threads 1", "--seed 1", "--nobootstrap",  
sep = "")
```

```
system(cur.command)
```

```
# load in the current ARACNe3 subnetwork
```

```
cur.subnetwork.data <- read.table(file = paste(cur.output.dir, "nobootstrap_network.txt",  
                                             sep = "/"), header = TRUE, sep = "\t")
```

```
# extract the edge names and write them to the output directory
```

```
cur.subnetwork.edge.values <- paste(as.character(cur.subnetwork.data$Regulator),  
                                   as.character(cur.subnetwork.data$Target), sep = "---")
```

```
saveRDS(cur.subnetwork.edge.values, file = paste(cur.output.dir, "/aracne3_subnetwork_",  
                                                cur.subnetwork.fold.num, "_edges.rds", sep = ""))
```

```
# add the edges to the compiled aracne edge counts
```

```
sub.loop.edge.values <- cur.subnetwork.edge.values
```

```
sub.loop.inter.edge.values <- intersect(sub.loop.edge.values, names(compiled.edge.counts))
```

```
sub.loop.diff.edge.values <- setdiff(sub.loop.edge.values, sub.loop.inter.edge.values)
```

```
compiled.edge.counts[match(sub.loop.inter.edge.values, names(compiled.edge.counts))] <-  
(1 +
```

```
compiled.edge.counts[match(sub.loop.inter.edge.values, names(compiled.edge.counts))])
```

```
sub.loop.diff.counts <- rep(1, times = length(sub.loop.diff.edge.values))
```

```
names(sub.loop.diff.counts) <- sub.loop.diff.edge.values
```

```

compiled.edge.counts <- c(compiled.edge.counts, sub.loop.diff.counts)

# check if the stopping criteria is met (i.e. all regulators have
# at least 50 targets)
cur.regulator.count.values <- sapply(strsplit(x = names(compiled.edge.counts),
                                             split = "---", fixed = TRUE), function(x) {
  return(x[1])
})

cur.regulator.count.values <- factor(cur.regulator.count.values, levels = sub.regulator.names)
cur.regulator.count.values <- table(cur.regulator.count.values)

cur.stop.check.value <- (mean(as.numeric(cur.regulator.count.values) >=
                               50) == 1)

}

```

```

# extract the regulator and target names from the final compiled
# ARACNe3 network data
cur.regulator.values <- strsplit(x = names(compiled.edge.counts), split = "---",
                                fixed = TRUE)

cur.target.values <- unlist(lapply(cur.regulator.values, function(x) {
  return(x[2])
})))

cur.regulator.values <- unlist(lapply(cur.regulator.values, function(x) {
  return(x[1])
})))

```



```

    return(z)
  })

sub.network.data$mi.values <- as.numeric(sub.mi.values)

return(sub.network.data)
})

names(cur.network.data.list) <- sub.regulator.names

# compute the Association Weight and Association Mode between each
# regulator and its target genes
aracne3.network.data.list <- lapply(cur.network.data.list, function(sub.data) {
  new.data <- sub.data[order(sub.data$count.values, sub.data$mi.values,
                           decreasing = TRUE), ]
  new.data$aw.values <- seq(from = nrow(new.data), to = 1, by = -1)/nrow(new.data)
  new.data$am.values <- new.data$scc.values
  return(new.data)
})

saveRDS(object = aracne3.network.data.list, file = paste(cur.output.dir,
                                                         "/TCGA_LUAD_", ncol(cur.network.tumor.copula.mat),
                                                         "_ARACNe3_Final_Network_List.rds",
                                                         sep = ""))

aracne3.df <- do.call("rbind", aracne3.network.data.list) %>% as.data.frame()
write.table(aracne3.df, file = 'data/aracne3.network.tfg.txt', row.names = FALSE)

```