```
## Kelly Botero - Juan Henao - Daniel Osorio

## INFORMACIÓN SUPLEMENTARIA:
## Comparación de redes de co-expresión derivadas de la interacción
## compatible e incompatible entre *Phythophthora infestans* y *Solanum phureja*.

# Cargando Librerías
library("GEOquery")
library("seqinr")
library("DESeq")
library("igraph")

# Descargando datos
# getGEO(GEO = "GSE11781",GSEMatrix = TRUE,destdir = ".")

# Leyendo Datos
GSE11781<-read.csv(gzfile("GSE11781_series_matrix.txt.gz"),
                   comment.char = "!",
                   sep = "\t",
                   stringsAsFactors = FALSE)
dim(GSE11781)[1]
```

```
## [1] 7625793
```

```
# Eliminando filas con datos incompletos
C_GSE11781<- GSE11781[complete.cases(GSE11781),]
dim(C_GSE11781)[1]
```

```
## [1] 78169
```

```
# Creando multifasta para BLAST
# write.fasta(sequences = strsplit(as.vector(C_GSE11781[1:dim(C_GSE11781)[1],1]),""),
#             names = 1:dim(C_GSE11781)[1],
#             file.out = "Solanum_phureja.fasta")

# BLAST remoto con cobertura 100% y E-Value menor a 0.05
# system("blastn -db nr -query Solanum_phureja.fasta
#         -task 'blastn-short' -perc_identity 100 -num_alignments 1
#         -outfmt 6 -evalue 0.05 -out Solanum_phureja.out -remote
#         -entrez_query 'txid4113[ORGN]'")

# Leyendo los datos de anotación
A_GSE11781<-read.csv("Solanum_phureja.out",sep = "\t",header = FALSE)

# Buscando genes únicos
names<-as.vector(A_GSE11781[,2])
IDs <- function(name){strsplit(name,"|",fixed = TRUE)[[1]][4]}
A_GSE11781[,2]<-as.vector(sapply(names,IDs))
T<-names(table(A_GSE11781[,2]))

# Declarando función para obtener el número máximo de conteos
colMax <- function(data){sapply(data, max, na.rm = TRUE)}
```

```r
# Función para extraer el número máximo de conteos desde nuestros datos
maxcounts<- function(id){
  colMax(C_GSE11781[A_GSE11781[A_GSE11781[,2]==id,1],2:7])
}

# Extrayendo datos finales
GSE11781<-t(sapply(T[grep("_",T)], maxcounts))
head(GSE11781)
```

```
##                GSM298187 GSM298188 GSM298189 GSM298190 GSM298191 GSM298192
## NM_001287831.1      1534      1150      1222       582       724       655
## NM_001287833.1        10         6         9         4         2         5
## NM_001287851.1        54        22        19        11        26        33
## NM_001287852.1        94        41        71        26        56        50
## NM_001287854.1        55        38        19        19        23        20
## NM_001287860.1       440      1294       344       401       274       348
```

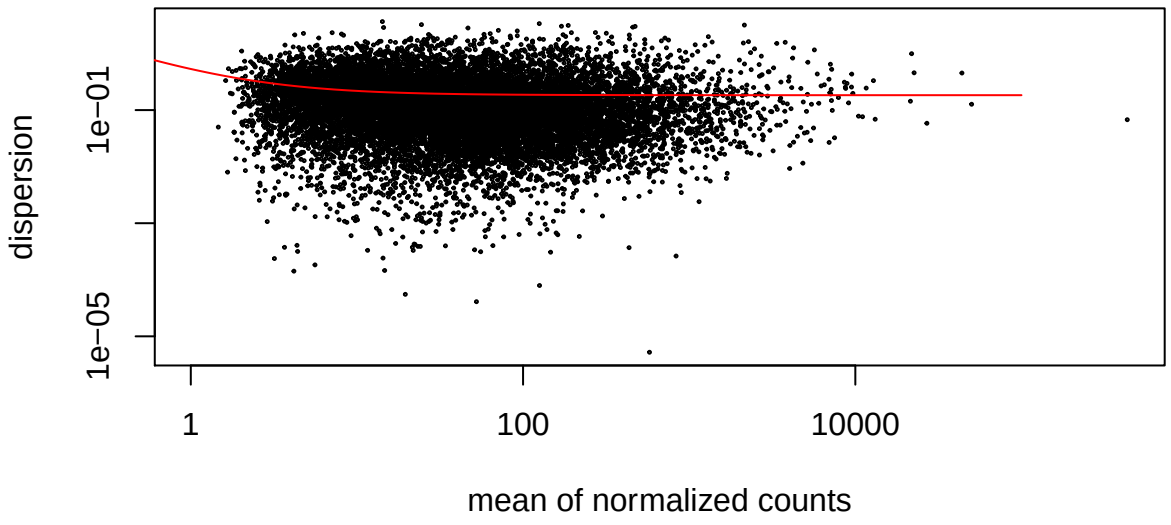
```r
# Escribiendo archivo de resultados
# write.table(x = GSE11781,
#             quote = FALSE,
#             sep = "\t",
#             file = "Solanum_phureja.anotado")

# Creando el countData
CD_GSE11781<-newCountDataSet(GSE11781, factor(c("R1","S1","R2","S2","R3","S3")))

# Normalizando datos
CD_GSE11781<-estimateSizeFactors(CD_GSE11781)
sizeFactors(CD_GSE11781)
```

```
## GSM298187 GSM298188 GSM298189 GSM298190 GSM298191 GSM298192
## 1.5609873 1.2305456 1.0976944 0.7489405 0.8270433 0.8286156
```

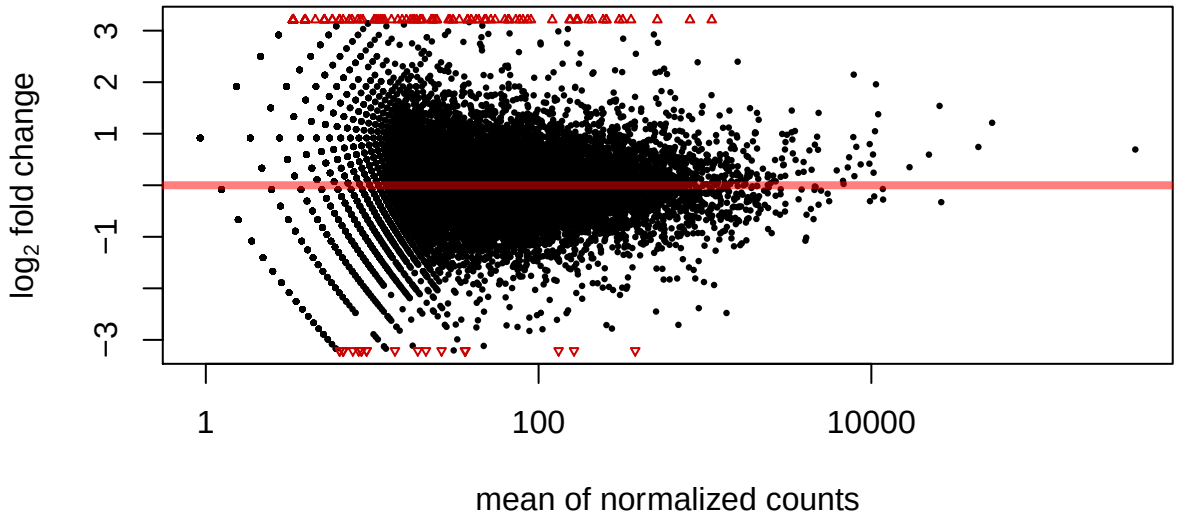
```r
# Imprimiendo datos normalizados
head(counts(CD_GSE11781,normalized=TRUE))
```

```
##                 GSM298187   GSM298188   GSM298189  GSM298190  GSM298191
## NM_001287831.1 982.711371  934.544794 1113.242411 777.097756 875.407604
## NM_001287833.1   6.406202    4.875886    8.199003   5.340878   2.418253
## NM_001287851.1  34.593490   17.878248   17.309006  14.687415  31.437290
## NM_001287852.1  60.218298   33.318554   64.681024  34.715707  67.711085
## NM_001287854.1  35.234110   30.880611   17.309006  25.369171  27.809910
## NM_001287860.1 281.872883 1051.566055  313.384116 535.423024 331.300668
##                 GSM298192
## NM_001287831.1 790.475079
## NM_001287833.1   6.034161
## NM_001287851.1  39.825462
## NM_001287852.1  60.341609
## NM_001287854.1  24.136644
## NM_001287860.1 419.977599
```

```
# Declarando la función para imprimir la dispersión
plotDispEsts <- function( datacds ){
  plot(rowMeans(counts( datacds, normalized=TRUE )),
       fitInfo(datacds)
       $perGeneDispEsts, cex=0.2, log="xy", ylab="dispersion",
       xlab="mean of normalized counts")
  xg = 10^seq( -.5, 5, length.out=300)
  lines( xg, fitInfo(datacds)$dispFun(xg), col="red" )
}

# Calculando dispersión con correcciones para una unica replica
CD_GSE11781<-estimateDispersions(CD_GSE11781,method='blind',sharingMode="fit-only")
plotDispEsts(CD_GSE11781)
```




```
## INTERACCIÓN INCOMPATIBLE
# Calculando diferencialmente expresados en interacción incompatible
RDE_GSE11781 <- nbinomTest(CD_GSE11781,"R1","R3")

# Definiendo expresión diferencial en interacción incompatible
RDE<-(quantile(abs(RDE_GSE11781$log2FoldChange[is.finite(RDE_GSE11781$log2FoldChange)]),
               probs = 0.99)*1.1)

# Graficando diferencialmente expresados en interacción incompatible
plotMA(RDE_GSE11781,col = ifelse(abs(RDE_GSE11781$log2FoldChange)>RDE,"red3","black"))
```



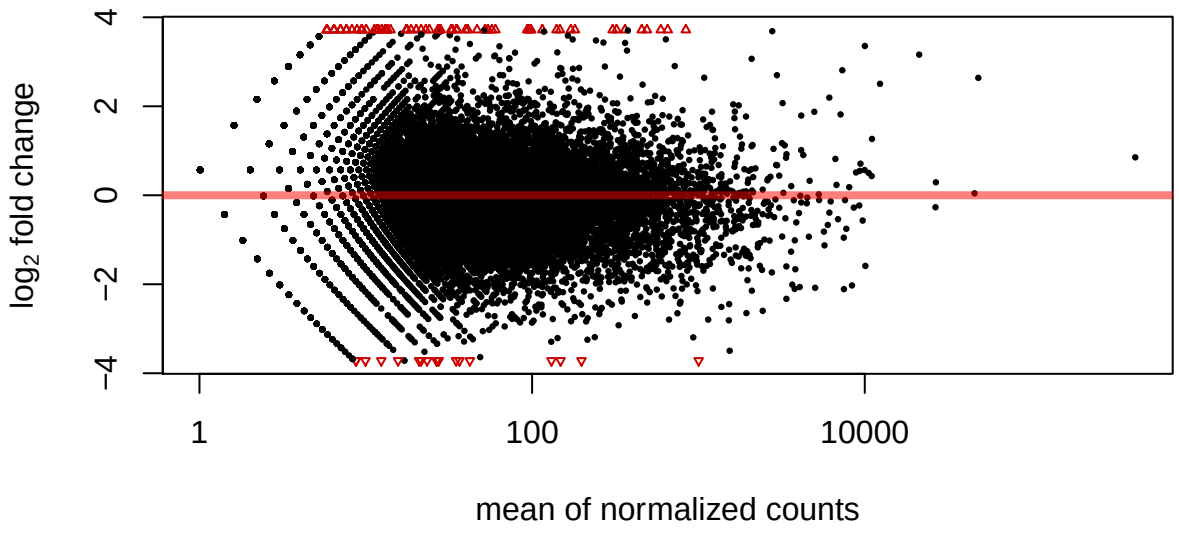

```
# Imprimiendo diferencialmente expresados en interacción incompatible
RDE_GSE11781<-RDE_GSE11781[abs(RDE_GSE11781$log2FoldChange)>RDE,]
dim(RDE_GSE11781)[1]
```

```
## [1] 120
```

```
head(RDE_GSE11781)
```

```
##                  id   baseMean  baseMeanA baseMeanB foldChange
## 81  NM_001288001.1  17.568392  1.2812404  33.85554  26.424037
## 130 NM_001288107.1  62.594528 11.5311634 113.65789   9.856585
## 197 NM_001288228.1 208.826724  8.9686827 408.68477  45.567982
## 272 NM_001288376.1 244.563868  0.6406202 488.48712 762.522200
## 278 NM_001288388.1  30.584530  1.9218606  59.24720  30.828043
## 676 XM_006338401.1   5.761379  0.6406202  10.88214  16.986881
##     log2FoldChange         pval         padj
## 81        4.723779 1.179606e-03 4.359259e-01
## 130       3.301088 2.248893e-03 7.423940e-01
## 197       5.509949 1.750612e-06 4.293000e-03
## 272       9.574636 2.794796e-10 1.599182e-06
## 278       4.946171 1.569471e-04 1.036213e-01
## 676       4.086349 3.384156e-02 1.000000e+00
```

```
## INTERACCIÓN COMPATIBLE
# Calculando diferencialmente expresados en interacción compatible
SDE_GSE11781 <- nbinomTest(CD_GSE11781,"S1","S3")

# Definiendo expresión diferencial en interacción compatible
SDE<-(quantile(abs(SDE_GSE11781$log2FoldChange[is.finite(SDE_GSE11781$log2FoldChange)]),
               probs = 0.99)*1.1)

# Graficando diferencialmente expresados en interacción compatible
plotMA(SDE_GSE11781,col = ifelse(abs(SDE_GSE11781$log2FoldChange)>SDE,"red3","black"))
```




```
# Imprimiendo diferencialmente expresados en interacción compatible
SDE_GSE11781<-SDE_GSE11781[abs(SDE_GSE11781$log2FoldChange)>SDE,]
dim(SDE_GSE11781)[1]
```

```
## [1] 87
```

```
head(SDE_GSE11781)
```

```
##                  id  baseMean baseMeanA  baseMeanB   foldChange
## 16  NM_001287879.1  20.12196  1.625295  38.618630  23.76099281
## 127 NM_001288103.1  93.54163  2.437943 184.645324  75.73816459
## 163 NM_001288167.1 170.82745 12.189715 329.465185  27.02812933
## 290 NM_001288424.1 321.64505  4.875886 638.414224 130.93297082
## 502 XM_006338107.1  23.35755 45.508268   1.206832   0.02651897
## 721 XM_006338476.1  57.33667  2.437943 112.235393  46.03692358
##     log2FoldChange         pval        padj
## 16        4.570523 9.819436e-04 0.337120874
## 127       6.242949 1.199591e-06 0.005148045
## 163       4.756390 1.682383e-05 0.041729238
## 290       7.032685 1.451870e-08 0.000249228
## 502      -5.236832 9.548923e-04 0.334560521
## 721       5.524720 1.819289e-05 0.041729238
```

```
# Identificando si existe un core de genes entre interaccion compatible e incompatible
SDE_GSE11781[SDE_GSE11781[,1] %in% RDE_GSE11781[,1],1]
```

```
## [1] "XM_006341347.1" "XM_006343755.1" "XM_006346404.1" "XM_006351206.1"
## [5] "XM_006353447.1" "XM_006355639.1" "XM_006366231.1" "XM_006366642.1"
## [9] "XM_006366683.1"
```

```
write.table(SDE_GSE11781[SDE_GSE11781[,1] %in% RDE_GSE11781[,1],1],
            quote = FALSE,row.names = FALSE,col.names = FALSE,
            file = "Genes_Core.txt")
```

```
# Matriz de similitud interacción compatible
M_SDE<-(abs(cor(t((counts(CD_GSE11781,normalized=TRUE)[SDE_GSE11781[,1],])),
                use = "p",method = "p")))

# Matriz de similitud interacción incompatible
M_RDE<-(abs(cor(t(counts(CD_GSE11781,normalized=TRUE)[RDE_GSE11781[,1],]),
                use = "p",method = "p")))


# Función que calcula valor umbral entre una red aleatoria y una red real
threshold<-function(smatrix){

  # Identifica el tamaño de la matriz
  n=nrow(smatrix)

  # Crea un vector de umbrales a ser evaluados:
  ltaos=seq(0.01,0.99,by=0.01)

  # Crea un vector que guarda el grado de nodo de cada gen (ki) y
  # los coeficientes de agrupamiento locales (Ci) por cada valor de tao:
  C<-K<-matrix(nrow=n, ncol=length(ltaos))

  # Calcula el grado de nodo (ki) y el coeficiente de
  # agrupamiento local (Ci) por cada valor de tao:
  for(tao in ltaos){

    # Matriz de adyacencia:
    A=matrix(0,nrow=n,ncol=n)

    # Completa la matriz de adyacencia usando la funci?n de adyacencia:
    for(i in 1:n){
      A[which(smatrix[,i]>=tao),i]<-1
      A[which(smatrix[,i]<tao),i]<-0
    }

    # Transforma la matriz A en un objeto igraph:
    A=graph.adjacency(A,mode="undirected",diag=FALSE)

    # Calcula el Ci de los nodos:
    Cv=transitivity(A,type="local")

    # Calcula el ki de los nodos:
    Kv=degree(A,loops=FALSE)
    ##Guarda Ci y ki en los vectores C y K respectivamente:
    K[,round(tao*100,0)]<-Kv
    C[,round(tao*100,0)]<-Cv
  }

  # Calcula el coeficiente de agrupamiento de la red (Co) y el coeficiente de agrupamiento
  # esperado para una red aleatoria (Cr), a distintos valores de tao:

  # Define vectores que guardan los coeficientes:
  Cr=Co=rep(0,100)
```

```
  # Para cada valor de tao:
  for(i in round(ltaos*100,0)){

    # Posición de los genes conectados en la red
    gn<-which(K[,i]>=1)

    # Número de nodos en la red
    kn=length(gn)

    # Variable en ecuación 3 (Ver Elo et.al. 2007)
    k1=1/kn*sum(K[gn,i])

    # Variable en ecuación 3 (Ver Elo et.al. 2007)
    k2=1/kn*sum(K[gn,i]^2)

    # Coeficiente de agrupamiento esperado para una red aleatoria
    Co[i]=((k2-k1)^2)/(kn*k1^3)

    # Si no hay nodos conectados: Co=0
    if(kn==0){Co[i]=0}

    # Posición de los genes con k1>1.
    gn<-which(K[,i]>1)

    # Número de genes con más de una arista en la red
    kn=length(gn)

    # Coeficiente de agrupamiento observado en la red.
    Cr[i]=1/kn*sum(C[gn,i])

    # Si no hay nodos conectados: Cr=0
    if(kn==0){Cr[i]=0}
  }

  # Función para suavizar la curva
  dif=runmed(abs(Cr-Co),k=3,endrule="constant")[1:100]

  # Función para identificar el primer máximo local
  localMaxima <- function(x) {
    y <- diff(c(-Inf, x)) > 0L
    rle(y)$lengths
    y <- cumsum(rle(y)$lengths)
    y <- y[seq.int(1L, length(y), 2L)]
    if (x[[1]] == x[[2]]) {
      y <- y[-1]
    }
    return (y)
  }
  # Retorna el primer máximo local
  return(ltaos[min(localMaxima(dif))])
}
# Calculo del Threshold por Red
threshold(M_RDE)
```

```
## [1] 0.77
```

```
threshold(M_SDE)
```

```
## [1] 0.94
```

```
#Finalmente se obtiene la matriz de adyacencia de la red al umbral seleccionado.
amatrix<-function(smatrix){

  #Define matriz de adyacencia:
  A=matrix(0,nrow=nrow(smatrix),ncol=nrow(smatrix))

  #Completa la matriz de adyacencia usando la función de adyacencia:
  A[which(smatrix>threshold(smatrix))]<-1

  #Agrega nombres a filas y columnas
  colnames(A)<-rownames(A)<-rownames(smatrix)

  #Convierte la diagonal en ceros (red no dirigida):
  diag(A)<-0

  #Elimina nodos no conectados:
  conected <- which(rowSums(A)>1)
  A=A[conected,conected]

  #Crea objeto igraph:
  A=graph.adjacency(A,mode="undirected",add.colnames=NULL,diag=FALSE)
  return(A)
}

#Guarda listado de aristas:
write.graph(amatrix(M_SDE),"Red_Sensible.txt",format="ncol")
write.graph(amatrix(M_RDE),"Red_Resistente.txt",format="ncol")
```