00) Funzioni

Mattia Manna

2024-12-09

Indice

Introduzione

Script che contiene tutte le funzioni utilizzate.
Vengono create queste funzioni per semplificare i passaggi successi e rendere la pipeline più efficace e comprensibile.

Queste funzioni fanno uso del pacchetto **pryr** ed in particolare della funzione *unenclose()* contenuta in esso. Questa funzione viene utilizza per obbligare le funzioni a non utilizzare l'environment generale nel quale sono, in questo modo hanno un comportamento più simile alle funzioni python e risultano meno prone ad errori.

1 Funzione per il file 01

1.1 distinct_patients, funzione di importazione

Si definisca una funzione per estrarre dai dataset delle conte dei geni, ad esempio:

- tep.expr.filtered
- tep.expr

solo i pazienti di una determinata condizione, ad esempio *Breast* cancer o *CRC* cancer.

Questa funzione prende in input:

- Il dataset dal quale estrarre i pazienti di una determinata condizione
- La condizione desiderata
- Il dataset contenente le informazioni per ogni sample. Nota che questo dataset deve essere del tipo **patients** con l'organizzazione vista dal [Gene Expression Omnibus](https://www.ncbi.nlm.nih.gov/geo/).

```
distinct_patients <- unenclose(function(df,cancer_type,patients){

  # Estrarre i nomi dei pazienti (es GSM1662534) con il tipo di cancro specificato
  specific.patients <- rownames(patients[patients$cancer.type.ch1==cancer_type,])

  # Filtrare il dataset in input prendendo solo
  output <- df[,colnames(df) %in% specific.patients]

  # Filtrare le righe dove la somma per riga non è uguale a zero, scartare geni inutili.
  #output <- output[rowSums(output != 0) > 0, ]
  return(output)
}
)
```

2 02 Identificazione DEGs

2.1 find_DEGs, pipeline per identificazione DEGs con edgeR

La funzione **find_DEGs** prende in input:

- **counts**: dataset contenente il livello trascrizione dei geni. Generalmente composto da due dataset uniti, ad esempio gbm + sani.
- **group**
- **contrasto**: per specificare il caso che si vuole studiare.
- **robust**: booleano. Se impostare la stima delle dispersione robust o no. Di default il valore è FALSE
- **prior.count**: di default impostare pari a 2.
- **logdue**, se restituire il $log_2()$ di ogni valore. Di default è FALSE.

La funzione **find_DEGs** restistuisce in output:

- La tep.expr.filtr normalizzata tramite metodo TMM
- Le stime
- Un oggetto contenente tutte le metriche

Il funzionamento della funzione in sintesi:

1. Crea un **oggetto DGEList** prendendo in input *counts* e *group*
2. **Normalizza** tramite il metodo **TMM**
3. Definisce la **design matrix**
4. Calcola le stime
5. Calcola le metriche come logCPM, logFC, FDR, PValue...
6. Restituitsce le metriche, i valori di espressione dei geni normalizzati, le stime

La parte fondamentale di questa funzione sono le **metriche**, una volta che si hanno le metriche basta fissare delle threshold per identificare i DEGs.

```
find_DEGs <- unenclose(function(counts,group,contrasto,robust,prior.count,logdue,nomi.geni = "NULL"){



  # Creazione dell'oggetto DGEList
  y <- edgeR::DGEList(counts = counts, group = group)


  # Normalizzazione
  y <- edgeR::calcNormFactors(y, method = "TMM")

  # Estrarre i geni normalizzati.
```

```
  # normalized.lib.sizes = TRUE omesso perchè TRUE di default
  # logical, if TRUE then log2 values are returned. (smooth the data)

  normalized_counts <- as.data.frame(edgeR::cpm(y,prior.count=prior.count,log = logdue))



  # build the design matrix
  design <- model.matrix(~ 0 + group)
  colnames(design) <- levels(group)

  # compute common, trend, tagwise dispersion
  y <- edgeR::estimateDisp(y,design = design,robust = robust)

  # fit the negative binomial GLM for each tag
  fit <-edgeR:: glmFit(y, design=design)

  # Definire il contrasto
  contrast <- limma::makeContrasts(contrasts=contrasto,levels = colnames(design))

  # Test statistico
  lrt <- edgeR::glmLRT(fit,contrast = contrast)

  if (length(nomi.geni) > 1 ){
    rownames(normalized_counts) <- nomi.geni

  }

  output <- list( tep.expr.filtr.normalized = normalized_counts,
                  stime = y,
                  tests = lrt

                )
  return(output)
})
```

2.2 degs_extraction, estrazione DEGs

Questa funzione prende in input l'output della funzione **find_DEGs**, ovvero *find_DEGSoutput*, e restituisce:

- Dataframe delle metriche, valori di logCPM, logFC, FDR, PValue per ogni gene
- Dataframe delle metriche, valori di logCPM, logFC, FDR, PValue solo per i DEGs
- DEGs.tep.expr.filtr, equivalente a tep.expr.filtr ma solo con DEGs
- DEGs.tep.expr.filtr.normalized, equivalente a tep.expr.filtr.normalized ma solo con DEGs

```
degs_extraction <- unenclose(function(find_DEGSoutput,logCPM.threshold,FDR.threshold,nomi.geni = "NULL"
  # Definire le threshold e identificare i DEGs

  ## Numero di geni presi in considerazione
  ngeni <- nrow(find_DEGSoutput$tep.expr.filtr.normalized)

  tep.expr.filtr.normalized <- find_DEGSoutput$tep.expr.filtr.normalized


  ## Estrarre l'oggetto che contiene le metriche per tutti gli n geni
  metriche <- edgeR::topTags(find_DEGSoutput$tests, n = ngeni)$table

  if (length(nomi.geni) > 1 ){
    rownames(metriche) <- nomi.geni

  }




  ## Applicare le threshold e trovare i DEGs
  DEGs.metriche <- metriche[(metriche$logCPM > logCPM.threshold)
                                        &
                                        (metriche$FDR < FDR.threshold), ]

  ## Printare numero di DEGs identitificati con le threshold scelte
  print("Quanti DEGs?")
  print(dim(DEGs.metriche)[1])

  DEGs.names <- rownames(DEGs.metriche)

  #-------------------------------------------------------------------------#
  # Estrarre i DEGs da  tep.expr.filtr.normalized

  ## Isolare  pancancer-hc DEGs nel tep.expr.filtr.normalized
  DEGS.tep.expr.filtr.normalized <- tep.expr.filtr.normalized[
                                                    rownames(tep.expr.filtr.normalized)
                                                     %in%
                                                     DEGs.names, ]

  print("Dimensioni del dataset tep.expr.filtr.normalized composto solo da DEGs")
```

```
  print(dim(DEGS.tep.expr.filtr.normalized))
  #-------------------------------------------------------------------------#



  output <- list( metriche = metriche,
                  DEGs.metriche = DEGs.metriche,
                  DEGS.tep.expr.filtr.normalized = DEGS.tep.expr.filtr.normalized,
                  DEGs.names = DEGs.names )


  return(output)
})
```

2.3 Volcano plot

Si vedano due funzioni che sfruttano il dataframe delle metriche e dei threshold su due metriche FC e Pvalue per detrminare quali siano gli UP e quali siano i DOWN regulated genes e poi produrre un volcano plot.

2.3.1 get_expr.table, funzione per creare la expr.table poi utilizzata per il volcano plot

La funzione **get_expr.table** prende in input:

- Dataframe delle metriche, valori di logCPM, logFC, FDR, PValue per ogni gene
- Threshold per FC
- Threshold per Pvalue

La funzione **get_expr.table** restistuisce in output:

- Restituisce la expr.table.

La expr.table è un dataframe che per ogni gene contiene le metriche per ogni gene e se quest'ultimo sia un DEGs UP o DOWN regulated o non sia affatto un DEGs.

```
get_expr.table <- unenclose(function(expr.table,fc_threshold,p_value_threshold){

  # Inizializzare tutti i geni a non differentially expressed
  expr.table$diffexpressed <- "NO";

  # Correggere i PValue per coerenza con il metodo DESeq2,
  ## va fatto questo perchè poi saranno confrontati
  expr.table$PValue.adjust <- stats::p.adjust( expr.table$PValue, method="fdr")


  # If the values of the genes are bigger then the FC threshold and smaller
  # than the p-value (null hypothesis not rejected)
  # then their are UP genes (upregulated genes)
  expr.table$diffexpressed[expr.table$logFC >= fc_threshold &
                             expr.table$PValue.adjust <= p_value_threshold] <- "UP"

  # If they are smaller than the FC threshold and smaller and smaller
  # than the p-value (null hypothesis not rejected)
  # They are DOWN (down regulated genes)
  expr.table$diffexpressed[expr.table$logFC  <= -fc_threshold &
                             expr.table$PValue.adjust <= p_value_threshold] <- "DOWN"

  expr.table$diffexpressed <- as.factor(expr.table$diffexpressed)

  return(expr.table)
})
```

2.3.2 volcano_plot, funzione per produrre un volcano plot

La funzione **volcano_plot** prende in input:

- Il dataframe expr.table

La funzione **volcano_plot** restistuisce in output:

- Restituisce nulla, produce solo il volcano plot

```
volcano_plot <- unenclose(function(expr.table,fc_threshold,p_value_threshold){
  library(ggplot2)
p <- ggplot2::ggplot(data=expr.table, ggplot2::aes(x=logFC, y=-log10(PValue.adjust), col=diffexpressed)
      ggplot2::geom_point() +
      ggplot2::xlab("fold change (log2)") +
      ggplot2::ylab("-log10 adjusted p-value") +
      ggplot2::geom_hline(yintercept=-log10(p_value_threshold), col="red")+
      ggplot2::geom_vline(xintercept=fc_threshold, col="red")+
     ggplot2::geom_vline(xintercept=-fc_threshold, col="red")

  print(p)


})
```

3 03 Pipeline differential co-expression network analysis

Dopo aver calcolato i **DEG** generalmente si procede con i **differential co-expression network** analysis in modo da identificarne gli hubs e restringere ancora il campo di quelli che possono essere i **potenziali biomarker**.

La pipeline per il calcolo dei differential co-expression network analysis prevede diverse funzioni.

Le principali sono:

- **evaluate_zscores**, utilizzata per il calcolo degli zscores
- **differential_zoexpression_network**, utilizzata per calcolare la adjacency.matrix, trovare hubs, costruire il network.

La funzione **evaluate_zscores** prende in input l'espressione normalizzata dei DEGs per le due patologie che si stanno confrontando.
Per poi procedere con:

1. Calcolo delle correlazioni
2. Applicare la fisher Transformation formula
3. Calcolare e restituire in output gli zscores

Più precisamente nella funzione **differential_coexpression network** (prende in input zscores e threshold):

1. Viene calcolata la adjacency matrix sfruttando gli zscores ed il threshold fornit
2. Viene calcolato il degree per ogni nodo
3. Vengono scartati i gene con degree pari a 0 sia dalla ricerca degli hubs sia dalla adjacency matrix
4. Vengono identificati gli hubs
5. Viene costruito il grafo sfruttando la adjacency matrix

3.1 evaluate_zscores, funzione per il calcolo per gli zscores

La funzione **evaluate_zscores** prende in input:

- Le tep.expr.filtr.normalized per le due condizioni

La funzione **evaluate_zscores** restistuisce in output:

- Restituisce gli zscores

```
#correlation with method Pearson
corr <- unenclose(function(dat){
  cor(t(dat),method = "pearson")
})

# Fisher Transformation formula
ft <- unenclose(function(x){
  return (1/2 * log((1+x)/(1-x)))
})

# Z-scores formula
z <- unenclose(function(x,y,n1,n2){
  v1 = 1/(n1-3)
  v2 = 1/(n2-3)
  zf = x - y
  vf = v1 + v2

  return (zf/sqrt(vf))
})

# Questa funzione non viene rinchiusa in un enclose() perchè fa utilizzo di
# funzioni esterne, tuttavia non utilizza alcun parametro esterno.
evaluate_zscores <- function(matrix.condition1,matrix.condition2){

  # similarity matrix condition 1
  sim.condition1 <- corr(matrix.condition1)

  # similarity matrix condition 2
  sim.condition2 <- corr(matrix.condition2)

  # remove diagonal
  diag(sim.condition1) <- diag(sim.condition1) <- 0
  diag(sim.condition2) <- diag(sim.condition2) <- 0


  # apply function to the respective matrices
  FT.sim.condition1 = apply(sim.condition1, 2, ft)
  FT.sim.condition2 = apply(sim.condition2, 2, ft)


  n1 <- ncol(matrix.condition1)
  n2 <- ncol(matrix.condition2)
```

```
  # apply function to the respective tumor types
  zscores <- mapply(z,as.data.frame(FT.sim.condition1),
                    as.data.frame(FT.sim.condition2),
                    n1,
                    n2
                   )

  rownames(zscores) <- colnames(zscores)

  return(zscores)
}
```

3.2 differential_coexpression_network, funzione per calcolare adjacency.matrix, trovare hubs, costruire il network

La funzione **differential_coexpression_network** prende in input:

- Gli zscores
- La threshold per gli zscores

La funzione **differential_coexpression_network** restistuisce in output:

- Un dataframe contenente gli hubs ed il loro degree
- Il differential co expression network
- La matrice di adiacenza
- Un dataframe contenente il degree di tutti i geni nel network
- Un dataframe contenent il degree di tutti i geni nel network escludendo quelli con degree pari a 0

```
differential_coexpression_network <-  unenclose(function(z,threshold,soglia_q=0.95){

  library(network)
  # Calcolare la adjacency.matrix basandosi sulla threshold fornita in input
  ## Se z >= t assegnare 1, sennò andare all'altro ifelse
  ## l'altro ifelse fa: se z <= -t assegnare 1 sennò 0
  adjacency.matrix <- ifelse(z > (threshold), 1, ifelse(z < (-threshold), -1, 0))

  #adjacency.matrix <- ifelse(z <= -threshold | z >= threshold, 1, 0)
  #diag(adjacency.matrix) <- 0

#-------------------------------------------------------------------------------#

  # Calcolare il degree per ogni gene
  degree <- rowSums(abs(adjacency.matrix))

  ## Creare un dataframe che fornisca il nome (enseble) per ogni gene ed il suo degree
  degree <- as.data.frame(cbind(colnames(adjacency.matrix),degree))

  ## Rinominare le colonne del dataframe per maggiore chiarezza
  colnames(degree) <- c("gene","degree")

  ## Trasformare la colonna degree da stringa in numeri interi
  degree$degree <- as.integer(degree$degree)

#-------------------------------------------------------------------------------#
  # Genes with a degree below 1 were excluded
  degree.zero <- degree[degree$degree > 0,]
#-------------------------------------------------------------------------------#

  # Hubs
  ## Calcolare la soglia per la quale un gene con degree superiore viene
  ## classificato come hubs
```

```
  q <- quantile(degree.zero$degree, probs = c (soglia_q),na.rm=T)

  ## Trovare gli hubs sfruttando la soglia q
  hubs <- degree.zero[degree.zero$degree > q, ]
#-------------------------------------------------------------------------------#


  # Escludere dalla adjacency.matrix i geni con degree pari a 0.

  ## Trovare il nome (enseble) dei geni che hanno degree pari a 0
  excluded.genes <- setdiff(degree$gene, degree.zero$gene)

  if (length(excluded.genes) != 0){
    ## Trovare l'indice (enseble) dei geni che hanno degree pari a 0
    idx <- which(colnames(adjacency.matrix) %in% excluded.genes)

    ## Rimuovere i geni  con degree pari a 0 sfruttando il loro  indice
    adjacency.matrix<- adjacency.matrix[-idx,-idx]
  }



  #diag(adjacency.matrix) <- 0
#-------------------------------------------------------------------------------#
  # Costruire il network utilizzando l' adjacency matrix
  net <- network::network(adjacency.matrix, matrix.type="adjacency",ignore.eval = T, directed = F)


  #net_igraph <-  graph_from_adjacency_matrix(adjacency.matrix, mode = "undirected", weighted = FALSE,

  # Converti la matrice dei pesi in un vettore di pesi per ogni arco presente
  #edge_list <- as_edgelist(net_igraph)  # Ottiene la lista degli archi (come coppie di nodi)
  #weights <- apply(edge_list, 1, function(x) z[x[1], x[2]])  # Usa la weight_matrix per assegnare i pe

  # Assegna i pesi agli archi
  #E(net_igraph)$weight <- weights


  #weighted.graph
  #weighted.graph <- network::network(z, matrix.type="adjacency", ignore.eval = F, directed = F)


#-------------------------------------------------------------------------------#
  #Definire l'output
  output <- list( all_degree = degree,
                  degree = degree.zero,
                  adjacency.matrix = adjacency.matrix,
                  network = net,
                  hubs = hubs,
                  geni.esclusi = excluded.genes
                  #igraph_network = net_igraph,
                  #weighted.graph = weighted.graph
                )
```

```
  return(output)
})

# Messo dopo permette di utilizzare le librerie
#differential_coexpression_network <- unenclose(differential_coexpression_network)
```

3.3 Informazioni sul grafo

Si costruisca una funzione costituita da diverse funzioni che restituisca grafici e informazioni sul differential co-expression network.

3.3.1 output_info, informazioni generali sull'output della funzione differential coexpression network

La funzione **output_info** prende in input:

- L' output della funzione **differential_coexpression_network**

La funzione **output_info** restistuisce in output:

- Non restituisce niente, printa solo delle informazioni sugli elementi contenuti nell'output

```
output_info <- unenclose(function(output){

  # Osservare l'output
  ## Dimensioni del dataframe contenente tutti i degs ed il loro degree nel grafo
  print("Dimensioni del dataframe contenente tutti i degs ed il loro degree nel grafo")
  print(dim(output$all_degree))

  ## Dimensioni del dataframe contenente tutti i degs con degree > 0 ed il loro degree
  ## nel grafo
  print("Dimensioni del dataframe contenente tutti i degs con degree > 0 ed il loro degree nel grafo")
  print(dim(output$degree))

  ## Dimensioni del dataframe contenente gli hubs
  print("Dimensioni del dataframe contenente gli hubs")
  print(dim(output$hubs))

  ## Dimensioni della adjacency matrix usata per definire il grafo
  print("Dimensioni della adjacency matrix usata per definire il grafo")
  print(dim(output$adjacency.matrix))

  # Quanti geni (nodes) ci sono nel grafo?
  print("Quanti geni (nodes) ci sono nel grafo?")
  print(network::network.size(output$network))

  # Quanti links (collegamenti tra geni) ci sono nel grafo?
  print("Quanti links (collegamenti tra geni) ci sono nel grafo?")
  print(network::network.edgecount(output$network))
})
```

3.3.2 plot_network, plottare il differential co-expression network

La funzione **plot_network** prende in input:

- L'oggetto R che contiente il differential co-expression network
- Il dataframe contenente la lista degli hubs ed il loro degree
- Un booleano chiamato *titoli*, che serve a decidere se avere un grafico con il titolo o meno

La funzione **plot_network** restistuisce in output:

- Non restituisce niente, printa solo il differential co-expression network tramite ggnet. In più evidenzia in rosso gli hubs, mentre gli altri geni sono in blu.

```
plot_network <- function(net,hubs,titoli){

  # DI questa non è possibile fare l'unenclose perchè c'è l'operatore  %v%

  library(ggnet)
  library(network)
  library(ggplot2)
  # Distinguere hubs e non
  net  %v% "type" = ifelse(network::network.vertex.names(net) %in% hubs$gene,
                           "hub", "non-hub")

  # Se il node è un hub assegnargli colore rosso, altrimenti blu
  net  %v% "color" = ifelse(net  %v% "type" == "hub",
                            "tomato", "deepskyblue3")


  # Creare l'oggetto ggnet
  p <- ggnet2(net,                    # Specificare il network da plottare
              color = "color",         # Usare i colori assegnati nell'attributo 'color'
              alpha = 0.7,             # Imposta il livello di trasparenza per i nodi del grafo.
                                       ## 0.7 -> nodi  leggermente trasparenti. 0 traspara
                                       ##  alpha tra 0 (traspararente) e 1 (opaco).
              size = 2,                # Dimensioni dei nodi

              edge.alpha = 1,          # Livello di trasparenza dei link
              edge.size = 0.15)  +     # Dimensioni dei link
              ggplot2::guides(size = "none") #Disattiva la legenda per la dimensione dei nodi.



  # Se titoli == T aggiungere il titolo al grafico, sennò no.
  if (titoli) {

    # Se si vuole un titolo (if soddisfatto) aggiungerlo
    p <- p  + ggtitle("Differential co-expression network ")  +  # Aggiungere il titolo
        theme(plot.title = element_text(size = 14, face = "bold", hjust = 0.5))  # Centra il titolo
  }
```

```
  # Plottare il grafo
  print(p)
}
```

3.3.3 plot_network_hubs, plottare il differential co-expression network composto da sole hubs

La funzione **plot_network_hubs** prende in input:

- La matrice di adiacenza
- Il dataframe contenente la lista degli hubs ed il loro degree
- Un booleano chiamato *titoli*, che serve a decidere se avere un grafico con il titolo o meno
- Il dataframe gene.info contenente le info per ogni gene

La funzione **plot_network_hubs** restistuisce in output:

- Non restituisce niente, plotta il differential co-expression network ma con focus esclusivamente tra gli hubs.

In questo modo si ha un grafo più piccolo e comprensibile.

```
plot_network_hubs <- unenclose(function(adjacency.matrix,hubs,titoli,gene.info,tipodataset="GEO"){


  library(ggnet)
  library(network)
  # Definire la adjacency matrix solo degli hubs
  ## Identificare l'indice degli hubs
  idx <- which(colnames(adjacency.matrix) %in% hubs$gene)

  ## Tramite l'indice definire una nuova  adjacency.matrix composta solo da hubs
  adjacency.matrix<- adjacency.matrix[idx,idx]



  #-------------------------------------------------------------------------------#

  # Creare un nuovo network composto solo da hubs
  net.hub <-network::network(adjacency.matrix, matrix.type="adjacency",ignore.eval = T, directed = F)

  # How many nodes (genes) in the graph?
  print("Nodes (grafo soli hubs):")
  print(network::network.size(net.hub))


  # How many links  in the graph?
  print("Links (grafo soli hubs):")
  print(network::network.edgecount(net.hub))

  if (tipodataset == "GEO"){
    # Crea le etichette dei nodi corrispondenti
    labels <- sapply(network::network.vertex.names(net.hub), function(x) {
      match_label <- gene.info$Symbol[gene.info$EnsemblGeneID == x]
      ifelse(length(match_label) > 0, match_label, NA)
    })
```

```
  }


  if (tipodataset == "TCGA"){
      labels <- sapply(network::network.vertex.names(net.hub), function(x) {
      match_label <- gene.info$gene_name[gene.info$gene_id == x]
      ifelse(length(match_label) > 0, match_label, NA)
    })

  }


  # Creare l'oggetto ggnet, che sarebbe il plot del grafo
  p<- ggnet::ggnet2(net.hub,
              color = "tomato", # Usare un colore unico per i nodi
              alpha = 0.9,
              size = 3,
              mode="fruchtermanreingold", # Specifica il layout del grafo.
                                          ## Il layout di Fruchterman-Reingold è un algoritmo
                                          ## di disposizione per grafi che cerca di ottimizzare
                                          ## la disposizione dei nodi
              edge.color = "lightgray",
              edge.alpha = 0.9,
              edge.size = 0.15,
              node.label = labels,  # Specifica le etichette
             label.color = "black", # Definisce il colore delle etichette
             label.size = 2) +  # Definisce la dimensione delle etichette
             ggplot2::guides(size = "none")



  # Se titoli == T verrà aggiunto il titolo, sennò no.
  if (titoli){

    # Se si vuole un titolo (if soddisfatto) aggiungerlo
    p <-   p  + ggplot2::ggtitle("Differential co-expression network focus sugli hubs") +   # Aggiunger
                ggplot2::theme(plot.title =  ggplot2::element_text(size = 14, face = "bold", hjust = 0.5)
  }



  print(p)
})
```

3.3.4 info_grafo, unire le 3 funzioni precedenti per richiamarle più facilmente

La funzione **info_grafo** prende in input:

- L' output della funzione **differential_coexpression_network**
- Un booleano chiamato *titoli*, che serve a decidere se avere un grafico con il titolo o meno Tramite titoli specificare se si vogliono o meno i titoli sui grafici.
  - titoli=**T**, ci sono i titoli.
  - titoli=**F**, non ci sono i titoli.

  Questa funzione è stata aggiunta perchè può essere utile se i grafici devono essere esportati su un documento latex ed il titolo verrò assegnato direttamente nel documento.
- Il dataframe gene.info contenente le info per ogni gene

La funzione **info_grafo** restistuisce in output:

- Non restituisce niente, si limita a richiamare le funzioni:
  - **output_info**
  - **plot_network**
  - **plot_network_hubs**

Questa funzione non fa uso di **unenclose()** perchè utilizza tutte funzioni esterne, tuttavia queste funzioni esterne utilizzano **unenclose()** quindi non ci sono problemi.

```
info_grafo <- function(output,gene.info,titoli,tipodataset="GEO"){

  # Restituire le dimensioni degli oggetti: adjacency matrix, hubs, network
  output_info(output)

  # Plottare tutto il grafo
  plot_network(output$network, output$hubs,titoli)

  # Plottare il grafo con focus sugli hubs
  plot_network_hubs(output$adjacency.matrix,output$hubs,titoli,gene.info,tipodataset)
}
```

3.3.5 info_hubs, funzione per mettere insieme le varie informazioni sugli hubs

Questa funzione mette insieme le informazioni di vari dataframe come:

- **hubs** per avere la lista degli hubs ed il loro degree
- **gene.info**, per le informazioni su i geni, il loro nome Symbol e non Ensemble, il ruolo che svolgono, descrizione, tipo...
- **expr.table** per avere le metriche FC, logCPM, FDR...

La funzione **info_hubs** prende in input:

- Il dataframe contenente la lista degli hubs ed il loro degree
- Dataframe delle metriche, valori di logCPM, logFC, FDR, PValue per ogni gene DEGs

La funzione **info_hubs** restistuisce in output:

- Un dataframe contenente tutte le informazioni reperibili per ogni gene DEGs
- Un dataframe contenente le informazioni più utili per ogni gene DEGs

```
info_hubs <- unenclose(function(hubs,dataframe.metriche,gene.info){
  hubs.info<- gene.info[gene.info$EnsemblGeneID %in%  hubs$gene, ]

  hubs.info <- merge(hubs,
                     hubs.info,
                     by.x = "gene",
                     by.y = "EnsemblGeneID",all=T )

  hubs.info <- merge(dataframe.metriche,
                     hubs.info,
                     by.x = "row.names",
                     by.y = "gene",all=F )

  colnames(hubs.info)[1] <- "EnsemblGeneID"

  hubs.shortinfo <- hubs.info

  ordine_colonne <- c(1,8,7,11,12,16,2,3,4,5,6,17)
  hubs.shortinfo <- hubs.shortinfo[,ordine_colonne]

  output <- list(all.hubs.info= hubs.info,
                 short.hubs.info = hubs.shortinfo
                )

  return(output)
})
```

3.4 compute_differential.co.expressed.network, differential co-expression network tramite sintassi di Sintassi del corso

Prende in innput

- Conta dei DEGs filtrata per la patologia 1, filtr.expr.deg.1
- Conta dei DEGs filtrata per la patologia 2, filtr.expr.deg.2

Restituisce:

- differential.co.expressed.network
- adjacency.matrix
- matrice.zscores

Questa funzione non fa uso di **unenclose()** perchè utilizza una funzione esterna.

```
transform_function <- unenclose(function(x) {
    log(x + 1, base=2)
  })

compute_differential.co.expressed.network <- function(filtr.expr.deg.1,
                                                      filtr.expr.deg.2){

  filtr.expr.deg.1 <- apply(filtr.expr.deg.1, c(1,2), transform_function)
  filtr.expr.deg.2 <- apply(filtr.expr.deg.2, c(1,2), transform_function)

  # Compute the correlation matrix for the LGG
  cor.mat.1 <- corr.test(t(filtr.expr.deg.1),
                         use = "pairwise",    # Utilizzare la correlazione di
                         method = "pearson", # pearson per individuare legami
                         adjust="fdr",ci=FALSE)

  # Compute the correlation matrix for the GBM
  cor.mat.2 <- corr.test(t(filtr.expr.deg.2),
                         use = "pairwise",    # Utilizzare la correlazione di
                         method = "pearson", # pearson per individuare legami
                         adjust="fdr",ci=FALSE)

  # Extract the correlation
  rho.1 <- cor.mat.1$r
  # The diagonals by construction will all be 1, fix at 0
  diag(rho.1) <- 0

  rho.2 <- cor.mat.2$r
  diag(rho.2) <- 0

  # Costruire il differential Co-expressed Network
  z1 <- 0.5*(log((1+rho.1)/(1-rho.1)))
  z2 <- 0.5*(log((1+rho.2)/(1-rho.2)))

  n1 <- ncol(filtr.expr.deg.1) - 3
```

```
n2 <- ncol(filtr.expr.deg.2) - 3


Z <- (z1-z2)/(sqrt((1/n1)+(1/n2)))
#t1 <- mean(Z) - 3*sd(Z)
#t2 <- mean(Z) + 3*sd(Z)
t1 <- -3
t2 <- 3
adj.mat.1_2 <- ifelse(Z <= t1 | Z >= t2, 1, 0)

net.1_2 <- network(adj.mat.1_2, matrix.type="adjacency",ignore.eval = FALSE, directed = F)

output <- list(grafo = net.1_2,
               adjacency.matrix = adj.mat.1_2,
               matrice.zscores = Z

               )

return(output)

}
```

3.5 Funzioni per interagire con il pacchetto diffcorr

Queste funzioni servono a manipolare gli output del pacchetto **diffcorr**.

3.5.1 from_diffcorr_to_zscores_matrix, calcolare gli zscores e creare una matrice che li contenga

Questa funzione prende in input:

- Output della funzione diffcorr
- Numero di samples per la prima condizione/patologia
- Numero di samples per la seconda condizione/patologia
- Dataset contenente i DEGs

Questa funzione:

- Calcola gli zscores prendendo le informazioni contenute nell'output della procedura diffcorr **comp.2.cc.fdr**.
- Inserire gli zscores in una matrice per poi avere un oggetto più facilmente convertibile in una adjacency matrix

```
from_diffcorr_to_zscores_matrix <- unenclose(function(out.diffcorr,
                                             n_samples_first_condition,
                                             n_samples_second_condition,
                                             DEGsconditions){

  # Ottenere gli zscores

  n1 <- n_samples_first_condition
  n2 <- n_samples_second_condition

  denominatore <- sqrt((1/ (n1-3)) +  (1/ (n2-3)))

  out.diffcorr$z1 <- (1/2) * log( (1+ out.diffcorr$r1) / (1- out.diffcorr$r1))
  out.diffcorr$z2 <- (1/2) * log( (1+ out.diffcorr$r2) / (1- out.diffcorr$r2))
  out.diffcorr$Z  <- (out.diffcorr$z1 - out.diffcorr$z2)/denominatore

  #----------#----------#----------#----------#----------#----------#----------#
  #----------#----------#----------#----------#----------#----------#----------#
  # Passare da un oggetto  **DiffCorr** ad una  matrice di zscores


  # Estrarre i nomi dei geni e calcolare la lunghezza
  genes.names <- rownames(DEGsconditions)
  n.adj <- length(genes.names)

  # Inizializzare la matrice di adiacenza con 0 sulla diagonale e NA altrove
  zscores.diffcorr <- matrix(NA,  # inizializzare la matrice con tutti NA values
                             nrow = n.adj, # numero di righe pari a n.adj
```

```
                            ncol = n.adj, # numero di colonne pari a n.adj

                            # Assegnare i nomi alle righe e alle colonne
                            dimnames = list(genes.names, genes.names)
                           )

  # Fissare tutti i valori della diagonale pari a zero.
  # Si fa questo perchè i valori sulla diagonale saranno sempre pari a zero in
  # questo tipo di matrici
  diag(zscores.diffcorr) <- 0

  # Utilizzare la funzione match per evitare il ciclo for
  idx1 <- match(out.diffcorr$`molecule X`, genes.names)
  idx2 <- match(out.diffcorr$`molecule Y`, genes.names)

  # Assegnare direttamente i valori zscore alla matrice (versione vettorializzata)
  zscores.diffcorr[cbind(idx1, idx2)] <- out.diffcorr$Z
  zscores.diffcorr[cbind(idx2, idx1)] <- out.diffcorr$Z


  return(zscores.diffcorr)
} )
```

3.5.2 differential_coexpression_network_diffcorr

Questa funzione prende in input un oggetto diffcorr con **p.adjust.method local**

- restituire la matrice con tutte le metriche fdr,
- la matrice di adiacenza
- gli hubs
- il degree per ogni nodo

```
differential_coexpression_network_diffcorr <-  unenclose(function(out.diffcorr,matrix.condition1,FDR.th

  # Estrarre i nomi dei geni e calcolare la lunghezza
  genes.names <- rownames(matrix.condition1)
  n.adj <- length(genes.names)

  # Inizializzare la matrice di adiacenza con 0 sulla diagonale e NA altrove
  matrice.diffcorr <- as.data.frame(matrix(NA,  # inizializzare la matrice con tutti NA values
                            nrow = n.adj, # numero di righe pari a n.adj
                            ncol = n.adj, # numero di colonne pari a n.adj

                            # Assegnare i nomi alle righe e alle colonne
                            dimnames = list(genes.names, genes.names)
                           ))


  diag(matrice.diffcorr) <- 0
```

```r
  # Utilizzare la funzione match per evitare il ciclo for
  idx1 <- match(out.diffcorr$`molecule X`, genes.names)
  idx2 <- match(out.diffcorr$`molecule Y`, genes.names)

  # Assegnare direttamente i valori zscore alla matrice (versione vettorializzata)
  matrice.diffcorr[cbind(idx1, idx2)] <- out.diffcorr$`lfdr (difference)`
  matrice.diffcorr[cbind(idx2, idx1)] <- out.diffcorr$`lfdr (difference)`

  adj.diffcorr <- ifelse(matrice.diffcorr < FDR.threshold , 1, 0)


  # Calcolare il degree per ogni gene
  degree <- rowSums(abs(adj.diffcorr))

  ## Creare un dataframe che fornisca il nome (enseble) per ogni gene ed il suo degree
  degree <- as.data.frame(cbind(colnames(adj.diffcorr),degree))

  ## Rinominare le colonne del dataframe per maggiore chiarezza
  colnames(degree) <- c("gene","degree")

  ## Trasformare la colonna degree da stringa in numeri interi
  degree$degree <- as.integer(degree$degree)

#-------------------------------------------------------------------------------#
  # Genes with a degree below 1 were excluded
  degree.zero <- degree[degree$degree > 0,]
#-------------------------------------------------------------------------------#

  # Hubs
  ## Calcolare la soglia per la quale un gene con degree superiore viene
  ## classificato come hubs
  q <- quantile(degree.zero$degree, probs = c (soglia_q))

  ## Trovare gli hubs sfruttando la soglia q
  hubs <- degree.zero[degree.zero$degree > q, ]
#-------------------------------------------------------------------------------#


  # Escludere dalla adjacency.matrix i geni con degree pari a 0.

  ## Trovare il nome (enseble) dei geni che hanno degree pari a 0
  excluded.genes <- setdiff(degree$gene, degree.zero$gene)

  if (length(excluded.genes) != 0){
    ## Trovare l'indice (enseble) dei geni che hanno degree pari a 0
    idx <- which(colnames(adj.diffcorr) %in% excluded.genes)

    ## Rimuovere i geni  con degree pari a 0 sfruttando il loro  indice
    adjacency.matrix<- adj.diffcorr[-idx,-idx]
  }


  net.diffcor <- network::network(adj.diffcorr, matrix.type="adjacency",ignore.eval = T, directed = F)
```

```
#-------------------------------------------------------------------------------#
  #Definire l'output
  output <- list( all_degree = degree,
                  degree = degree.zero,
                  adjacency.matrix = adj.diffcorr,
                  network = net.diffcor,
                  hubs = hubs,
                  geni.esclusi = excluded.genes,
                  matricefdr = matrice.diffcorr

                )


  return(output)
})
```

4 Funzioni per la vecchia metodologia

4.1 DEGs tramite DESeq2

4.1.1 old_find_DEGs, funzione per calcolare i DEGs secondo la metodologia del corso

```
old_find_DEGs <- unenclose(function(full.data,metad,first_group_length,thresholds_list){

  #library(DESeq2)

  # Define the DESq2 object
  dds <- DESeq2::DESeqDataSetFromMatrix(countData=full.data,# Provide gene-patient data
                                colData=metad,      # Provide the condition of the patient
                                                  # patient - condition (LGG o GBM)
                                design= ~condition, # Specify the column
                                tidy=TRUE)

  total_rows <- ncol(full.data)-1


  # Questa parte di selezione dei geni comuni viene rimossa perchè
  # non c'è una parte simile in edgeR

  # Select the genes that are in at least the 90% of the patients
  #n_pazienti_90<-  round((total_rows/100) * 90,2)
  #keep <- rowSums(counts(dds) >= 10) >= n_pazienti_90

  ## Ridefine the dataframe by taking only the genes
  ## that are in at least the 90% of the patients
  #dds <- dds[keep,]

  #--------#

  # Normalize through DESq2
  dds <- DESeq2::estimateSizeFactors(dds)
  normalized_counts <- DESeq2::counts(dds, normalized=TRUE)



  filtr.expr.1 <- as.data.frame(normalized_counts[, 1:first_group_length])
  filtr.expr.2 <- as.data.frame(normalized_counts[, (first_group_length+1):total_rows])

  # Evaluate Fold Change (FC)
  fc <- log2(rowMeans(filtr.expr.1) / rowMeans(filtr.expr.2))

  # Rename the values with the name of the corrispondend gene
  names(fc) <- rownames(filtr.expr.1)

  # Get the  p-values
  pval.fc <- sapply(1:nrow(filtr.expr.1),
                    function(i)(t.test(filtr.expr.1[i,], filtr.expr.2[i,] ))$p.value)
```

```
  # Correcting for multiple comparison
  pval.fc.fdr <- p.adjust(pval.fc, method="fdr")

  # Put those values in a dataframe
  expr.table <- data.frame(cbind(fc, pval.fc.fdr))

  # Round the values of the FC
  expr.table[,1] <- round(expr.table[,1],2)




  for (i in 1:length(thresholds_list)){
    # Define the thresholds
    fc_threshold <- thresholds_list[[i]]$FC
    p_value_threshold <- thresholds_list[[i]]$p_value


    # Select only the genes that are differenitally expressed
    # By selecting the ones that are above the FC and below the p value
    deg.genes <- rownames(expr.table[abs(expr.table$fc) >= fc_threshold &
                                       expr.table$pval.fc.fdr <=p_value_threshold,])

    filtr.expr.deg.1 <-filtr.expr.1[rownames(filtr.expr.1) %in% deg.genes,]
    filtr.expr.deg.2 <-filtr.expr.2[rownames(filtr.expr.2) %in% deg.genes,]

    cat("FC:", fc_threshold, "& p-value:", p_value_threshold, "-> Number of differentially expressed ge
  }


  output <- list(expr.table= expr.table,
                 filtr.expr.normalized.1 = filtr.expr.1,
                 filtr.expr.normalized.2 = filtr.expr.2
              )

  return(output)
})
```

4.1.2 old_extract_deg_genes, funzione per selezione DEG genes per determinati threshold

```
old_extract_deg_genes <- unenclose(function(expr.table,fc_threshold,p_value_threshold){
   deg.genes <- rownames(expr.table[abs(expr.table$fc) >= fc_threshold &
                                      expr.table$pval.fc.fdr <=p_value_threshold,])

})
```

4.1.3 old_volcano_plot, funzione per volcano plot

```
old_volcano_plot <- unenclose(function(expr.table,fc_threshold,p_value_threshold){
  #library(ggplot2)
  # Inizializzare tutti i geni a non differentially expressed
  expr.table$diffexpressed <- "NO";

  # If the values of the genes are bigger then the FC threshold and smaller
  # than the p-value (null hypothesis not rejected)
  # then their are UP genes (upregulated genes)
  expr.table$diffexpressed[expr.table$fc >= fc_threshold &
                             expr.table$pval.fc.fdr <= p_value_threshold] <- "UP"

  # If they are smaller than the FC threshold and smaller and smaller
  # than the p-value (null hypothesis not rejected)
  # They are DOWN (down regulated genes)
  expr.table$diffexpressed[expr.table$fc <= -fc_threshold &
                             expr.table$pval.fc.fdr <= p_value_threshold] <- "DOWN"

  expr.table$diffexpressed <- as.factor(expr.table$diffexpressed)

  p <- ggplot2::ggplot(data=expr.table, ggplot2::aes(x=fc, y=-log10(pval.fc.fdr), col=diffexpressed))+
    ggplot2::geom_point() +
     ggplot2::xlab("fold change (log2)") +
     ggplot2::ylab("-log10 adjusted p-value") +
     ggplot2::geom_hline(yintercept=-log10(p_value_threshold), col="red")+
     ggplot2::geom_vline(xintercept=fc_threshold, col="red")+
     ggplot2::geom_vline(xintercept=-fc_threshold, col="red")

    print(p)
    return(expr.table)
})
```

5 Esportare funzioni

```
save(list = ls(), file = "/Users/mattia/Desktop/Università/Magistrale/Tesi/R/Comparazione metodologie/Da
```

```
## Warning in save(list = ls(), file =
## "/Users/mattia/Desktop/Università/Magistrale/Tesi/R/Comparazione
## metodologie/Data/00) Funzioni.RData"): 'package:pryr' may not be available when
## loading
```