GBM analysis

Mattia Manna

2025-01-13

Indice

```r
dataframe.formato.classificazione <- function(dataframe){

  # Trasporre il dataframe delle conte di test, in questo modo si hanno le conte
  # disposte nel modo giusto per la classificazione: pazienti sulle righe e geni sulle colonne
  dataframe <- as.data.frame(t(dataframe))

  # Aggiungere le label y (cancer sano) alle conte
  dataframe <- merge(dataframe,patients.summary,by.x="row.names",
                                    by.y="sample_name",all = FALSE)

  # Sistemare il dataframe dopo l'operazione di merge, rimettere i rownames al posto giusto
  rownames(dataframe) <- dataframe$Row.names

  # Eliminare la colonna rownames
  dataframe <- dataframe %>% select(-"Row.names")

  return(dataframe)
}
```

1 Importazione dati, pulizia e filtraggio

1.1 Importazione dati

Importare i dati salvati localmente.

```r
tep.expr <- read.delim("~/Desktop/Università/Magistrale/Tesi/R/GSE68086/GSE68086_TEP_data_matrix.txt", r
dim(tep.expr)
## [1] 57736   285

patients <- read.csv("~/Desktop/Università/Magistrale/Tesi/R/GSE68086/patients.csv", row.names=1, sep="
dim(patients)
## [1] 285  51

gene.info <- read.delim("~/Desktop/Università/Magistrale/Tesi/R/GSE68086/Human.GRCh38.p13.annot.tsv", ro
dim(gene.info)
## [1] 39376    17
```

1.2 Pulizia dati

Siccome ci sono delle incongruenze tra il nome dei sample nei dataframe:

- patients, dataset contenente le informazioni per ogni sample
- tep.expr, dataset contenente la conta per i geni

sistemare queste incongruenze.

```r
patients$sample_name <- sapply( patients$source_name_ch1,    # Vettore al quale applicare la funzione

                                # Funzione da applicare ad ogni valore nel vettore
```

```
                                function(x) gsub("-", ".", x)
                                )
colnames(tep.expr)<- sapply(colnames(tep.expr), function(x) sub("X", "", x))
```

1.3 Filtrare

Filtrare per la condizione: almeno un sample con count > 5

```
tep.expr.filtr <- tep.expr %>% filter_all(any_vars(. >= 5))
dim(tep.expr.filtr)
```

```
## [1] 16347   285
```

1.4 Prendere count GBM e SANI e filtrare di nuovo

```
# Specificare la condizione da analizzare
condizione.da.analizzare = "GBM"

# Estrarre MALATI con la condizione specificata
condition.samples <- patients[patients$cancer.type.ch1==condizione.da.analizzare,]$sample_name

# Dimensioni del samples (controllo per la correttezza dei risultati)
n.condition.samples <- length(condition.samples)
print(n.condition.samples)
## [1] 40



# Estrarre il nome/codice dei  sample SANI
healhty.samples <- patients[patients$cancer.type.ch1=="HC",]$sample_name

## Dimensioni del samples (controllo per la correttezza dei risultati)
length(healhty.samples)
## [1] 55


# Unire SANI e MALATI
tutti.samples <- c(condition.samples,healhty.samples)
length(tutti.samples)
## [1] 95


# Filtrare la matrice delle conte di nuovo perchè sono state selezionate solo alcune patologie
counts <- tep.expr.filtr[, tutti.samples]
counts <- counts %>% filter_all(any_vars(. >= 5))
dim(counts)
## [1] 14411    95
```

Estrarsi in sommario dei pazienti presi in considerazione.

```
patients.summary <- patients[patients$sample_name %in% colnames(counts),c("sample_name","cancer.type.ch
patients.summary[1:5,]
```

```
##               sample_name cancer.type.ch1
## GSM1662606      VU280.GBM             GBM
## GSM1662607 VU284.GBM.vIII             GBM
## GSM1662608 VU306.GBM.vIII             GBM
## GSM1662609   VU360.GBM.WT             GBM
## GSM1662610 VU372.GBM.vIII             GBM
```

2 Calcolo dei DEGS sul TRAIN set

2.0.1 DEGS su tutto il dataframe con TMM

Numero degs su tutto il dataframe senza TMM (no divisione train e test).

```
group <- factor(c(rep("GBM",40),rep("HC",55)))

# Creazione dell'oggetto DGEList
oggettoDGEList <- DGEList(counts = counts, group = group)

oggettoDGEList <- calcNormFactors(oggettoDGEList, method = "TMM")

# Funzione equivalente a calcNormFactors ma che permette di salvare informazioni aggiuntive sui parametri
# ad esempio: ref column, divisore = exp(mean(log(f)))
parametri.TMM <- TMMbyHand(counts)

# Normalizzare con CPM e le standard lib sizes
normalized_counts <- cpm(oggettoDGEList,
                         normalized.lib.sizes = T,
                         prior.count= 1 ,
                         log = T)

normalized_counts <- as.data.frame(normalized_counts) # renderlo un dataframe

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

colnames(design) <- levels(group)

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

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

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

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

logCPM.threshold <- 3
FDR.threshold <- 0.01
## Estrarre l'oggetto che contiene le metriche per tutti gli n geni
metriche <- topTags(lrt, n = nrow(counts))$table

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

```r
nomiDEGS <- rownames(DEGs.metriche)
dim(DEGs.metriche)
```

```
## [1] 860   5
```

```r
tep.expr.filtr.normDEGS <- normalized_counts[rownames(normalized_counts) %in%
                                                        nomiDEGS,]

tep.expr.filtr.normDEGS <- dataframe.formato.classificazione(tep.expr.filtr.normDEGS)

# Esportare il dataframe
write.csv(tep.expr.filtr.normDEGS,
          file = "/Users/mattia/Desktop/Università/Magistrale/Tesi/Python TMM/Data/GBM_DEGS_normalized_
          row.names =T )
```

2.0.2 Osservazione parametri di normalizzazione

Per prima cosa fare un check e controllare che i fattori di normalizzazione siano gli stessi.

```r
all.equal(as.numeric(parametri.TMM$fattori.normalizzazione),oggettoDGEList$samples$norm.factors)
```

```
## [1] TRUE
```

I fattori sono uguali tra le due procedure.
Si può procedere a salvare ed esportare i parametri.

```r
# Salva la lista in un file
save(parametri.TMM, file = "/Users/mattia/Desktop/Università/Magistrale/Tesi/R/Normalizzazione TMM/Para
```

2.1 Differential co-expressed network

```
DEGS.tep.expr.filtr.normalized.gbmVShc <- normalized_counts[rownames(normalized_counts) %in% rownames(DE
```

```
matrix.condition1 <- DEGS.tep.expr.filtr.normalized.gbmVShc[, condition.samples]
matrix.condition2 <- DEGS.tep.expr.filtr.normalized.gbmVShc[, healhty.samples]
Zscores.gbmVShc <- evaluate_zscores(matrix.condition1,matrix.condition2)

# Soglia
z_threshold <- 3

output.gbmVShc <- differential_coexpression_network(Zscores.gbmVShc,z_threshold)
```

```
## [1] 42  2
```

```
##                              gene degree
## ENSG00000012822 ENSG00000012822     68
## ENSG00000013016 ENSG00000013016    129
## ENSG00000026297 ENSG00000026297     58
## ENSG00000034152 ENSG00000034152     60
## ENSG00000090273 ENSG00000090273     97
## ENSG00000093167 ENSG00000093167     64
## ENSG00000096060 ENSG00000096060     58
## ENSG00000099783 ENSG00000099783    141
## ENSG00000100319 ENSG00000100319    125
## ENSG00000100911 ENSG00000100911     82
```

```
tep.expr.filtr.normHUBS <- normalized_counts[rownames(normalized_counts) %in%
                                                output.gbmVShc$hubs$gene,]

tep.expr.filtr.normHUBS <- dataframe.formato.classificazione(tep.expr.filtr.normHUBS)

# Esportare il dataframe
write.csv(tep.expr.filtr.normHUBS,
          file = "/Users/mattia/Desktop/Università/Magistrale/Tesi/Python TMM/Data/GBM_HUBS_normalized_
          row.names =T )
```

3 Metriche di centralità

```
matrice.adiacenza <- output.gbmVShc$adjacency.matrix
matrice.adiacenza <- abs(matrice.adiacenza)
```

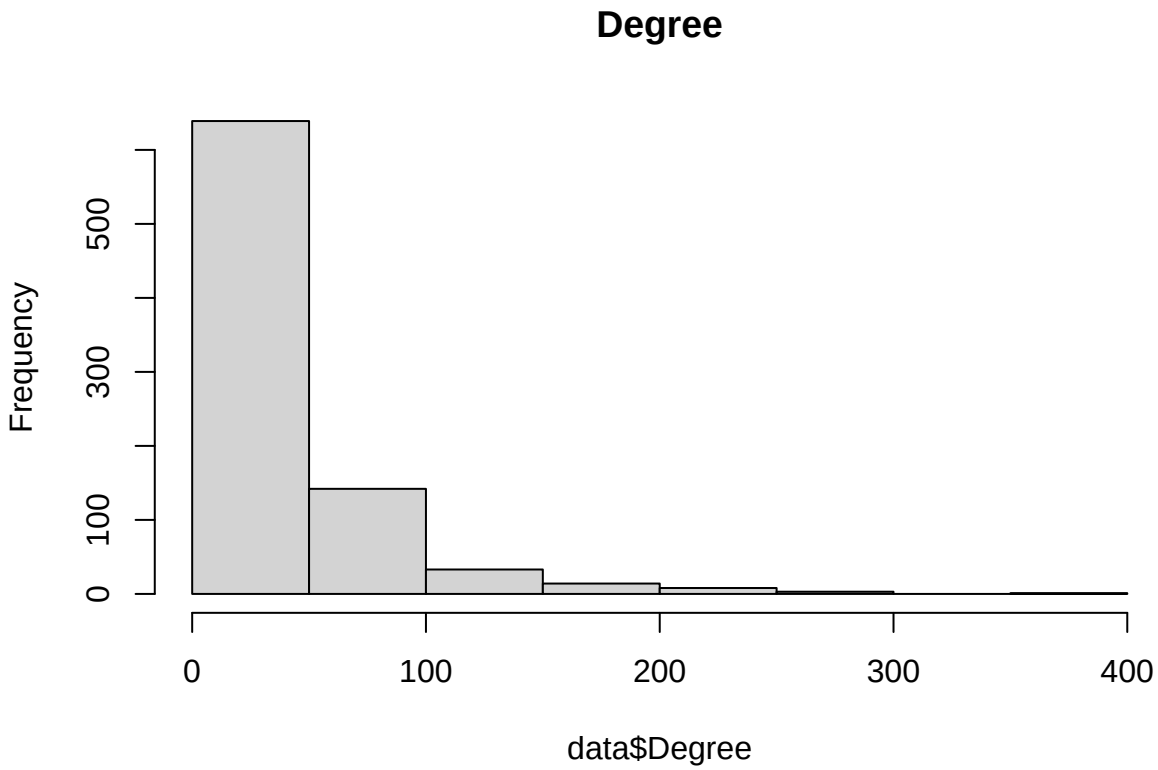
```
# Creare il network igraph dalla matrice di adiacenza
network <- graph_from_adjacency_matrix(matrice.adiacenza)

data <- data.frame(
  "Gene" = V(network)$name,
  "Degree" = degree(network),
  "betweenness_centrality"= betweenness(network),
  "closeness_centrality"=   closeness(network),
  "eigenvector_centrality" = eigen_centrality(network)$vector,
  "local_clustering_coefficient" =transitivity(network, type="local")
)
data[is.na(data)] <- 0
dim(data)
```

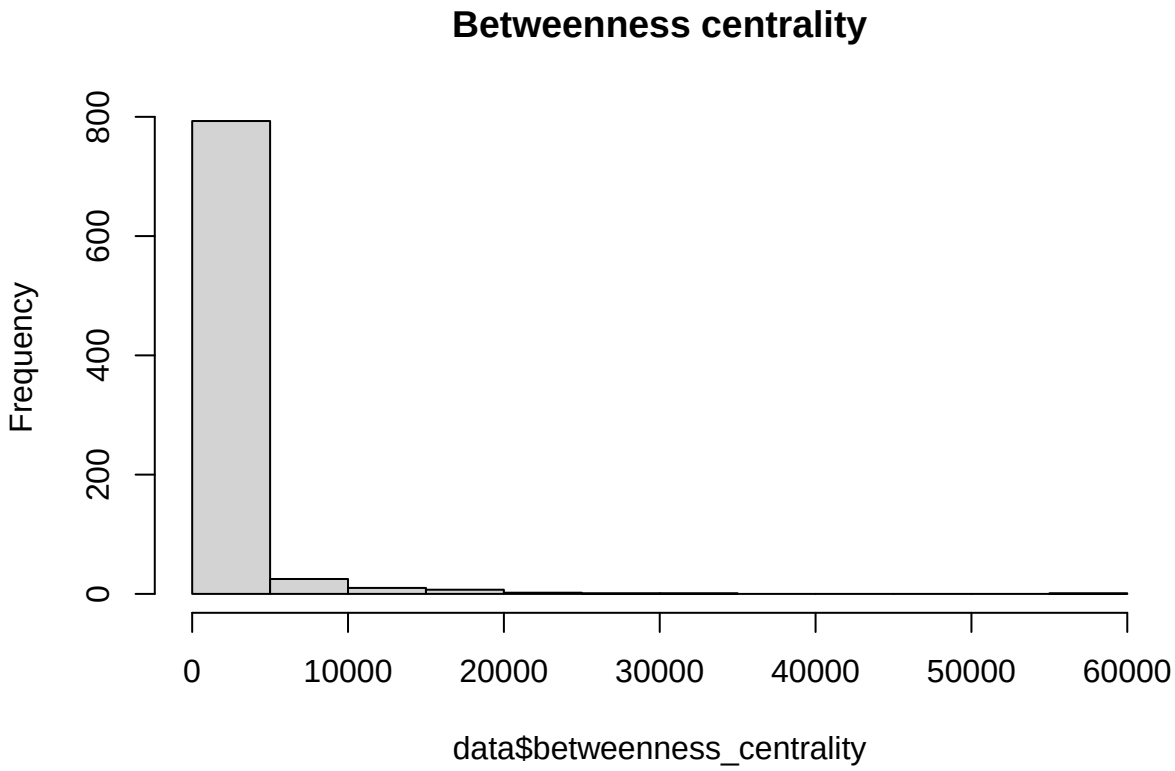
```
## [1] 840   6
```

3.1 Istogrammi

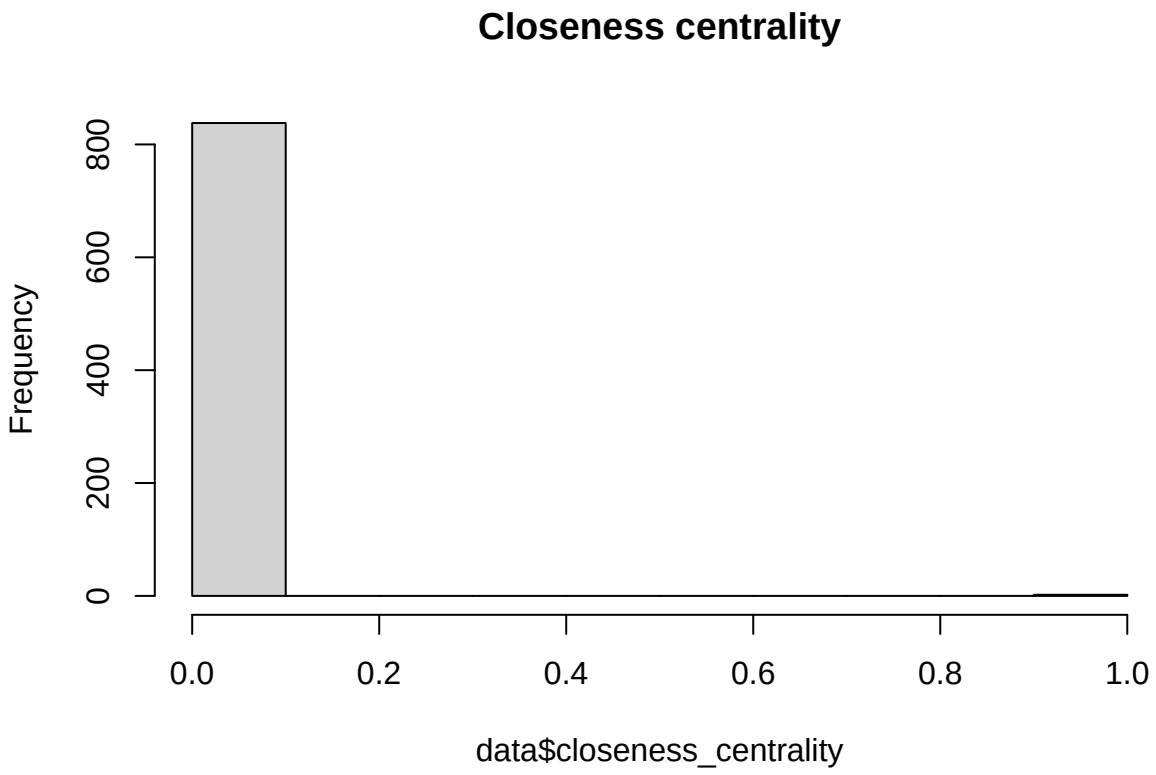
3.1.1 Degree



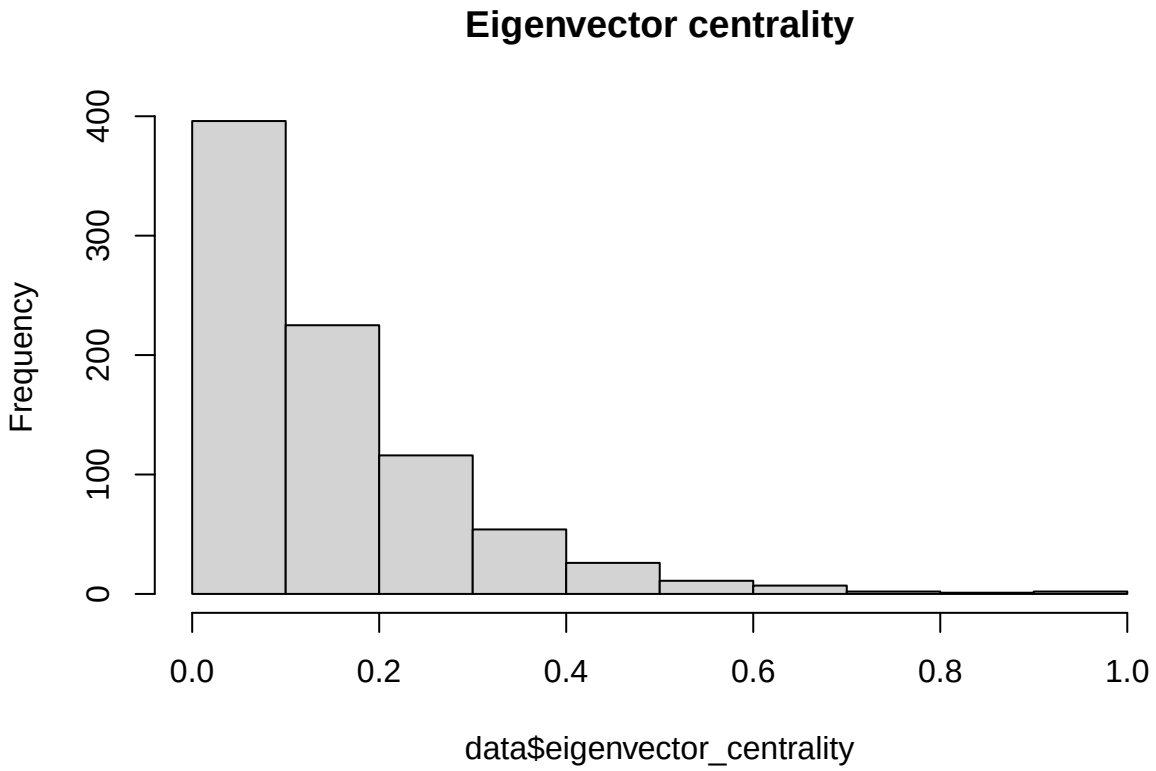
3.1.2 Betweenness centrality



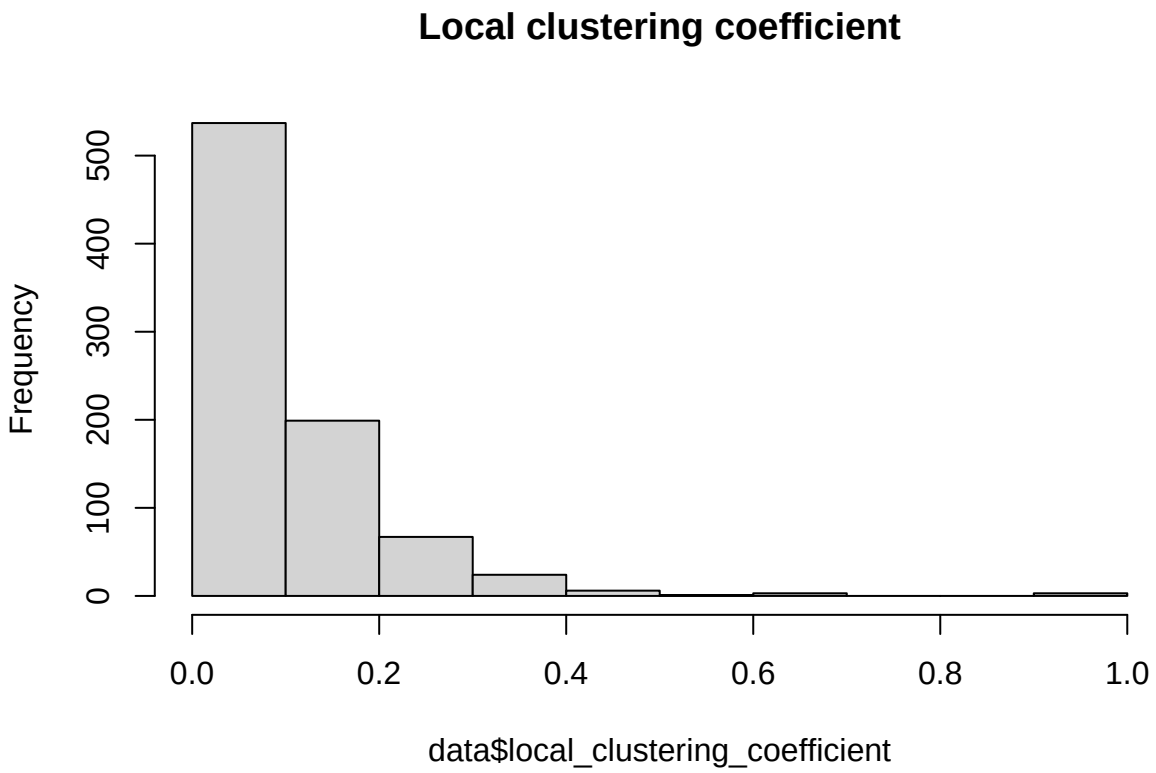
3.1.3 Closeness centrality



3.1.4 Eigenvector centrality



3.1.5 Local clustering coefficient



3.2 Betweenness centrality

```
soglia <- 0.95

metrica <- data$betweenness_centrality
q <- quantile(metrica, probs = c(soglia))

# Selezionare solo i geni con eigenvector_centrality superiore al 95% percentile
genes <- data[metrica > q,]$Gene

# Estrarre solo i geni con eigenvector_centrality superiore al 95% percentile
# dalla matrice delle conte filtrata e normalizzata
tep.expr.filtr.normBETWEENNESS <- normalized_counts[rownames(normalized_counts) %in%
                                                      genes,]
dim(tep.expr.filtr.normBETWEENNESS)
## [1] 42 95
nrow(tep.expr.filtr.normBETWEENNESS) # Numero geni selezionati tramite metrica
## [1] 42

tep.expr.filtr.normBETWEENNESS <- dataframe.formato.classificazione(tep.expr.filtr.normBETWEENNESS)
dim(tep.expr.filtr.normBETWEENNESS)
## [1] 95 43

# Esportare il dataframe
write.csv(tep.expr.filtr.normBETWEENNESS,
          file = "/Users/mattia/Desktop/Università/Magistrale/Tesi/Python TMM/Data/GBM_BETWEENNESS_norma
          row.names =T )
```

3.3 Closeness centrality

3.3.1 Ultimo 5% (0.95)

```
soglia <- 0.95

metrica <- data$closeness_centrality
q <- quantile(metrica, probs = c(soglia))

# Selezionare solo i geni con eigenvector_centrality superiore al 95% percentile
genes <- data[metrica > q,]$Gene

# Estrarre solo i geni con eigenvector_centrality superiore al 95% percentile
# dalla matrice delle conte filtrata e normalizzata
tep.expr.filtr.normCLOSENESS <- normalized_counts[rownames(normalized_counts) %in%
                                                    genes,]
dim(tep.expr.filtr.normCLOSENESS)
## [1] 42 95
nrow(tep.expr.filtr.normCLOSENESS) # Numero geni selezionati tramite metrica
## [1] 42

tep.expr.filtr.normCLOSENESS <- dataframe.formato.classificazione(tep.expr.filtr.normCLOSENESS)
dim(tep.expr.filtr.normCLOSENESS)
## [1] 95 43

# Esportare il dataframe
write.csv(tep.expr.filtr.normCLOSENESS,
          file = "/Users/mattia/Desktop/Università/Magistrale/Tesi/Python TMM/Data/GBM_CLOSENESS_normali
          row.names =T )
```

3.3.2 Primo 5 % (0.05)

Ha due code quindi considerare anche il primo 5%.

```
soglia <- 0.05
metrica <- data$closeness_centrality
q <- quantile(metrica, probs = c(soglia))
genes <- data[metrica < q,]$Gene
tep.expr.filtr.normCLOSENESS_last5 <- normalized_counts[rownames(normalized_counts) %in%
                                                    genes,]
dim(tep.expr.filtr.normCLOSENESS_last5)
## [1] 42 95
nrow(tep.expr.filtr.normCLOSENESS_last5) # Numero geni selezionati tramite metrica
## [1] 42
tep.expr.filtr.normCLOSENESS_last5 <- dataframe.formato.classificazione(tep.expr.filtr.normCLOSENESS_las
dim(tep.expr.filtr.normCLOSENESS_last5)
## [1] 95 43
# Esportare il dataframe
write.csv(tep.expr.filtr.normCLOSENESS_last5,
          file = "/Users/mattia/Desktop/Università/Magistrale/Tesi/Python TMM/Data/GBM_CLOSENESSlast5_n
          row.names =T )
```

3.4 Eigenvector centrality

```
soglia <- 0.95

metrica <- data$eigenvector_centrality
q <- quantile(metrica, probs = c(soglia))

# Selezionare solo i geni con eigenvector_centrality superiore al 95% percentile
genes <- data[metrica > q,]$Gene

tep.expr.filtr.normEIGEN <- normalized_counts[rownames(normalized_counts) %in%
                                                genes,]
dim(tep.expr.filtr.normEIGEN)
## [1] 42 95
nrow(tep.expr.filtr.normEIGEN) # Numero geni selezionati tramite metrica
## [1] 42

tep.expr.filtr.normEIGEN <- dataframe.formato.classificazione(tep.expr.filtr.normEIGEN)
dim(tep.expr.filtr.normEIGEN)
## [1] 95 43

# Esportare il dataframe
write.csv(tep.expr.filtr.normEIGEN,
          file = "/Users/mattia/Desktop/Università/Magistrale/Tesi/Python TMM/Data/GBM_EIGEN_normalized
          row.names =T )
```

3.5 Local Clustering coefficient

```
soglia <- 0.95

metrica <- data$local_clustering_coefficient
q <- quantile(metrica, probs = c(soglia))

# Selezionare solo i geni con eigenvector_centrality superiore al 95% percentile
genes <- data[metrica > q,]$Gene

tep.expr.filtr.normCLUSTER <- normalized_counts[rownames(normalized_counts) %in%
                                                genes,]
dim(tep.expr.filtr.normCLUSTER)
## [1] 37 95
nrow(tep.expr.filtr.normCLUSTER) # Numero geni selezionati tramite metrica
## [1] 37

tep.expr.filtr.normCLUSTER <- dataframe.formato.classificazione(tep.expr.filtr.normCLUSTER)
dim(tep.expr.filtr.normCLUSTER)
## [1] 95 38

# Esportare il dataframe
write.csv(tep.expr.filtr.normCLUSTER,
          file = "/Users/mattia/Desktop/Università/Magistrale/Tesi/Python TMM/Data/GBM_CLUSTER_normaliz
          row.names =T )
```