PROGETTO DATA MINING

GIORGIO LOCICERO

INTRODUZIONE

Durante questo studio tenterò di stabilire una strategia di analisi su dati che mira a scoprire relazioni tra pazienti afflitti da diverse patologie sulla base di distribuzioni di geni, create fittando dei campioni di individui a distribuzioni scelte secondo il test di Kolmogorov-Smirnov (la scelta sarà tra tre tipi di distribuzioni, weibull, normale e lognormale).

Per costruire una misura utile, calcoliamo la significatività statistica degli individui con le patologie per i diversi geni che si presentano nel record, in questo modo vediamo se il gene della persona appartiene effettivamente alla distribuzione creata e se l'ipotesi per cui segue quel tipo di distribuzione viene rispettata dal record.

Verranno calcolati allora due insiemi di dati, uno formato dall'area della coda destra della distribuzione di weibull associata al gene specifico e calcolata a partire dal valore che assume il record del paziente con patologia(p-value dell'ipotesi unilatera per cui $X_0 <= X$, dove X è una variabile aleatoria che rappresenta il valore atteso), verranno considerati valori di fiducia di α=0,05, il secondo insieme di dati è semplicemente un booleano(o alternativamente o 1 o 0) per ogni parametro appartenente al record, che esprime semplicemente l'accettazione dell'ipotesi descritta per il primo insieme di dati tramite il p-value.

I metodi di clusterizzazione utilizzati saranno principalmente gerarchici e di partizionamento(k-means e simili), per misurare la qualità dei cluster ottenuti useremo il coefficente di silhouette .

Durante questa analisi utilizzerò R per lavorare sui dati.

La visualizzazione dei dati sarà molto poca principalmente per il fatto che valutiamo punti in più dimensioni(quando calcoliamo le distanze tra i vari individui per la clusterizzazione) e in spazi non euclidei(quando abbiamo le ipotesi al posto dei geni).

Per la visualizzazione delle distribuzioni porterò un esempio per un singolo gene.

FITTING DI UN CAMPIONE

Come già specificato, le distribuzioni considerate per il fit dei dati saranno tre(Weibull, Normale e Lognormale), ogni singolo gene avrà una distribuzione diversa.

DISTRIBUZIONE DI WEIBULL

Tentiamo di costruire delle distribuzioni di weibull per ogni gene basandoci su un campione di popolazione.

La distribuzione di weibull distribuita secondo parametri $\alpha,\beta \in \mathrm{R}_{+}$, (scale e shape)viene definita secondo la seguente densità di probabilità:

$$f_X(t) = \begin{cases} 0, & \text{se t} < 0; \\ \alpha\beta t^{\beta-1}\exp\left(-\alpha t^{\beta}\right), & \text{se t} \geq 0; \end{cases}$$

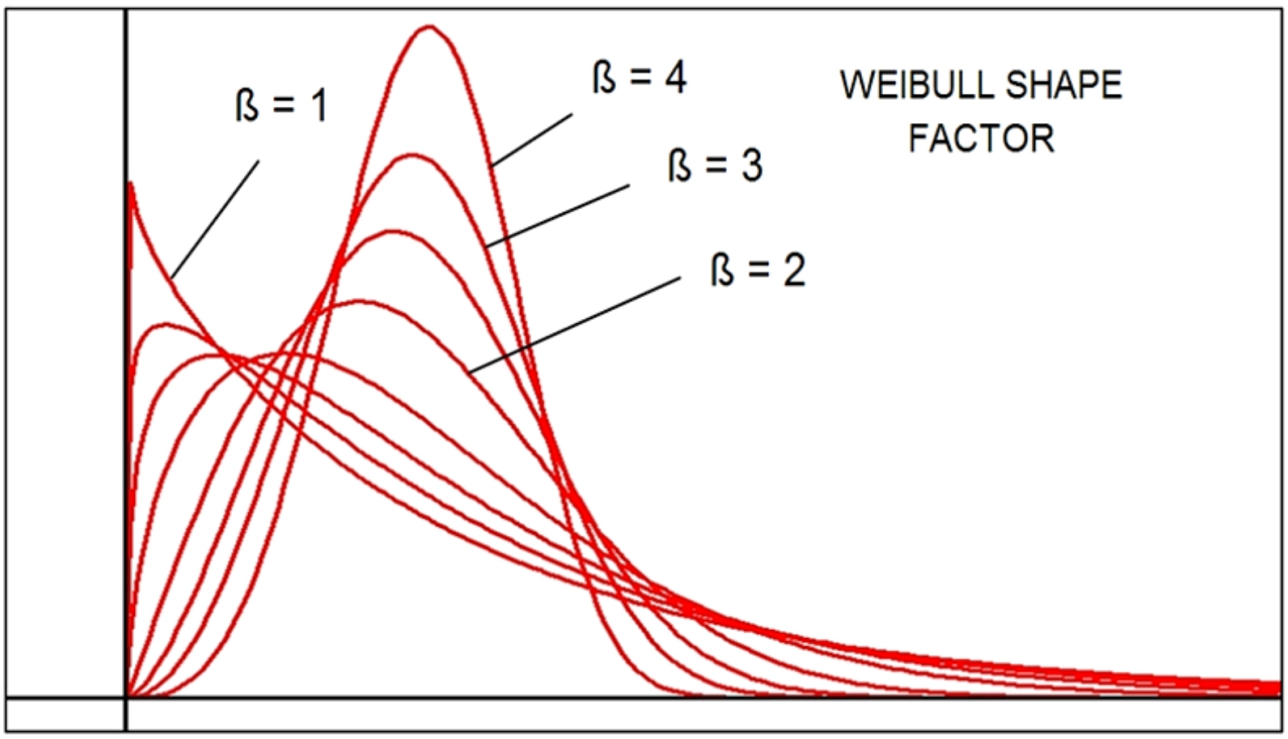
Da cui la funzione di ripartizione

$$F_X(t) = \begin{cases} 0, & \text{se t} < 0; \\ 1-\exp(-\alpha t^{\beta}), & \text{se t} \geq 0; \end{cases}$$

La media e la varianza di questo tipo di distribuzione sono le seguenti:

$$\mathbb{E}[X] = \alpha^{-\frac{1}{\beta}}\Gamma\left(1+\frac{1}{\beta}\right) \quad , \quad V[X] = \alpha^{-\frac{2}{\beta}}\left\{\Gamma\left(1+\frac{2}{\beta}\right) - \left[\Gamma\left(1+\frac{1}{\beta}\right)\right]^2\right\}$$

dove $\Gamma(z)$ è la funzione gamma di eulero.



I nostri dati sono formati da record per ogni individuo(colonna) formati a loro volta da geni specifici per ogni individuo.

Poichè i dati sono sia negativi che positivi, dobbiamo shiftare i valori in modo che siano tutti maggiori di 0 (sia la distribuzione di Weibull che quella log-normale richiedono dati positivi)

Per fittare i nostri dati con la distribuzione di weibull non utilizzerò la funzione *fitdistr* della libreria MASS che calcola i parametri di una distribuzione a partire dai dati che si vogliono analizzare, poichè questa funzione creà molti problemi per valori piccoli e molti valori.

Il codice della vecchia funzione per creare una matrice N x 2 (due parametri per N geni) è il seguente:

```
weibullpar <- function(M){
 retmat <- matrix(nrow = nrow(M),ncol = 2)
 for (i in 1:nrow(M)) {
  retmat[i,] <- fitdistr(M[i,],densfun = "weibull")$estimate
 }
 retmat
}
```

Una funzione che non da nessun problema è invece contenuta nella libreria **fitdistrplus**, più lenta della fitdistr normale, ma fitta meglio i dati e non da messaggi di errore senza alcuna spiegazione(oltre a dare altre misure importanti per l'analisi di goodness come **aic** o **bic** , ma non userò queste misure per scegliere il fit migliore).

```
weibullpar <- function(M){
  retmat <- matrix(nrow = nrow(M),ncol = 2)
  for (i in 1:nrow(M)) {
   retmat[i,] <- fitdist( M[i,] ,distr = "weibull",lower = c(0.01,0.01))$estimate
  }
  retmat
}
```

Questi dati sono stati shiftati secondo la seguente funzione:

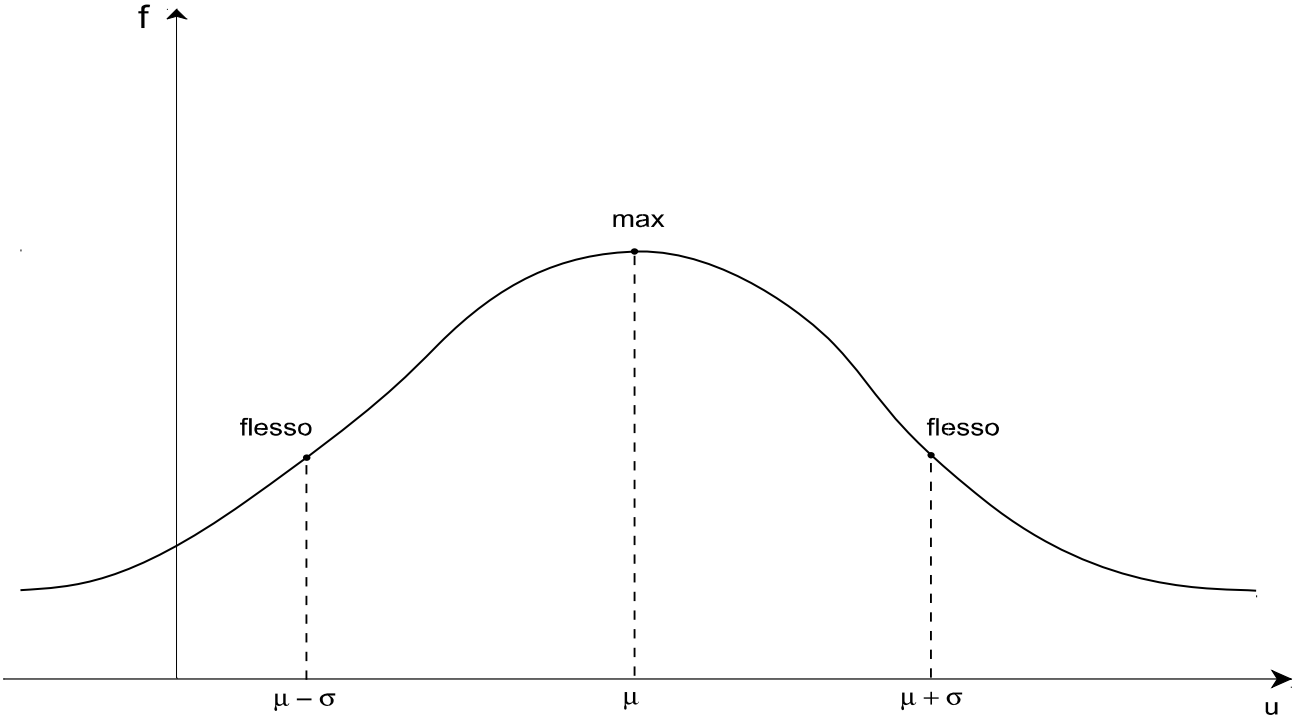
```
shifted_values <- function(M){
 retmat <- M
 for (i in 1:nrow(M)) {
  minim <- min(M[i,])
  if(minim<=0)retmat[i,] <- M[i,] - minim  +0.01
 }
 retmat
}
```

Il problema principale di questo ragionamento è che i valori di minimo non sono quelli della popolazione ma solo dei campioni, quindi nella eventualità in cui subentrino altri individui con valori negativi più grandi, questa analisi sarebbe inconcludente.

DISTRIBUZIONE NORMALE

Una variabile aleatoria X è detta distribuita secondo una normale di parametri $\mu \in \mathrm{R}$, $\sigma \in \mathrm{R}_+$ ovvero $X \sim N(\mu,\sigma)$ se ha densità di probabilità

$$f_X(u) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2}\frac{(u-\mu)^2}{\sigma^2}\right]$$



Distribuzione Normale - funzione $f_X(u)$

la distribuzione normale non ha bisogno che i dati siano shiftati a destra per essere tutti maggiori di 0.

Il codice in R per calcolare i parametri della distribuzione fittata è il seguente:

```
normalpar <- function(M){
  retmat <- matrix(nrow = nrow(M),ncol = 2)
  for (i in 1:nrow(M)) {
    retmat[i,] <- fitdistr(M[i,],densfun = "normal")$estimate
  }
  retmat
}
```

DISTRIBUZIONE LOGNORMALE

La variabile aleatoria

$$X = e^N$$

segue la distribuzione lognormale

$$\log \mathcal{X}(\mu, \sigma^2)$$

se e solo se

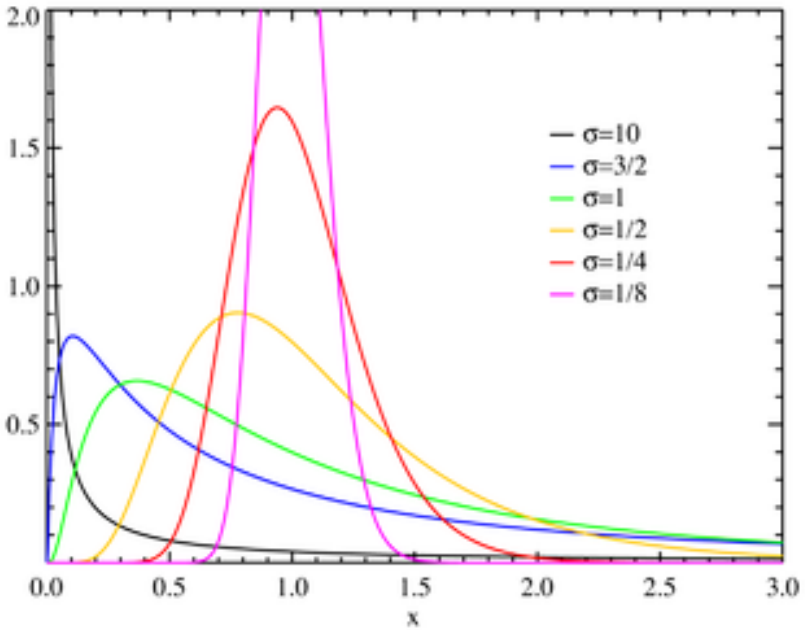
$$N = \log X$$

segue la distribuzione normale

$$\mathcal{N}(\mu, \sigma^2)$$

La sua funzione di densità di probabilità è (per $x > 0$)

$$f(x) = \frac{e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}}{x\sqrt{2\pi}\sigma}$$



Distribuzione lognormale - funzione $f(u)$

Questa distribuzione può essere usata solo per valori positivi, quindi deve essere shiftata come quella di weibull.

Il codice in R per fittare i nostri dati è il seguente:

```
lognormalpar <- function(M){
 retmat <- matrix(nrow = nrow(M),ncol = 2)
 for (i in 1:nrow(M)) {
  retmat[i,] <- fitdistr(M[i,],densfun = "lognormal")$estimate
 }
 retmat
}
```

Se si volesse adottare una strategia differente ai valori negativi, si potrebbero togliere quei valori per le singole distribuzioni(in modo da inalterare le distribuzioni fittate risultanti), però si dovrebbe considerare quanti siano questi valori negativi, e considerare un threshold dove per valori minori, si possono eliminare i record con valori negativi, mentre per valori maggiori, si shifta come già descritto la distribuzione cambiando radicalmente i parametri.

Il codice che svolge questa funzione per la distribuzione log-normale(ma il ragionamento è lo stesso per la weibull) è il seguente:

```
lognormalpar_enhanched <- function(M){
 minors_vect <- minors_row(M)
 retmat <- matrix(nrow = nrow(M),ncol = 2)
 for (i in 1:nrow(M)) {
  if(minors[i]>100){
   retmat[i,]<- fitdistr(M[i,] - min(M[i,]) +0.0001,densfun = "lognormal")$estimate
  }
  else{
   retmat[i,] <- fitdistr(M[i,M[i,]>0],densfun = "lognormal")$estimate
  }
 }
 retmat
}
```

Un'altra strategia in visione del clustering su parametri sarebbe quella di shiftare tutto quanto secondo il negativo minimo globale in modo che le distribuzioni siano tutte maggiori di 0 ed i parametri si possano confrontare senza ambiguità tra i vari geni(dato che shiftando singoli geni non si sa quanto sia effettivo il confronto tra geni data la loro dissimilarità)

Il codice che implementa questa strategia è il seguente:

```
lognormalpar_total <- function(M){
 global_min <- min(M)
 if(global_min<=0)M <- M -global_min +0.0001
 retmat <- matrix(nrow = nrow(M),ncol = 2)
 for (i in 1:nrow(M)) {
   retmat[i,] <- fitdistr(M[i,],densfun = "lognormal")$estimate
 }
 retmat
}
```

Utilizzeremo questa funzione e quella di weibull(dove le modifiche sono le stesse) per fittare le distribuzioni dato che i parametri ci serviranno per clusterizzare sulla base delle singole distribuzioni.

FITTING BASATO SULLA BONTÀ DELL'ADATTAMENTO

Per vedere quale sia la distribuzione che approssima meglio i dati useremo il test di Kolmogorov Smirnov per distribuzioni continue.

Una caratteristica interessante di questo test è che la distribuzione della statistica del test K-S stessa non dipende dalla funzione di distribuzione cumulativa sottostante testata. Un altro vantaggio è che si tratta di un test esatto (il test di bontà di adattamento chi-quadro dipende da una dimensione del campione adeguata affinché le approssimazioni siano valide). Altri test sono stati creati come miglioramenti del test di Kolmogorov-Smirnov(Anderson-Darling test e il Cramer Von-Mises test).

La statistica del test di Kolmogorov-Smirnov è definita come :

$$D = \max_{1 \le i \le N} \left(F(Y_i) - \frac{i-1}{N}, \frac{i}{N} - F(Y_i) \right)$$

dove F è la distribuzione cumulativa teorica della distribuzione sottoposta a test che deve essere una distribuzione continua.

L'ipotesi relativa alla forma distributiva viene respinta se la statistica del test, D, è maggiore del valore critico ottenuto da una tabella dove sono stabiliti dei valori standard, durante questo studio si userà questa statistica per stabilire la distribuzione migliore, prendendo la statistica minore. Alternativamente si può utilizzare il p-value ottenuto.

Il codice per fittare per avere la maggior bontà di adattamento è il seguente(considerando minimi locali per ogni gene):

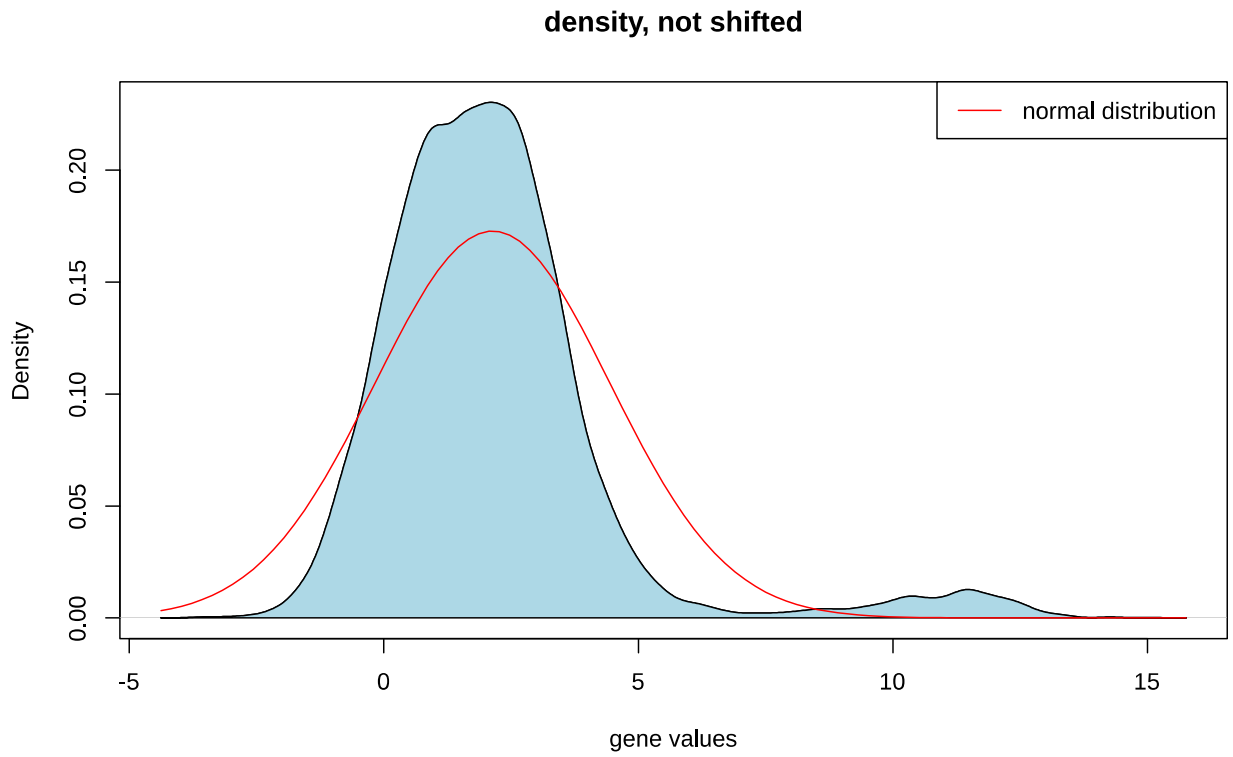
```
best.fit.ks2 <- function(M){
 shifted_values.genes <- shifted_values(M)
 parametri_weibull <- weibullpar(shifted_values.genes)
 parametri_normal <- normalpar(M)
 parametri_lognormal <- lognormalpar(shifted_values.genes)
 best.fit.ks <- parametri_lognormal
 outdistri <- rep("lognormal",nrow(parametri_lognormal))
 for (i in 1:nrow(parametri_lognormal)) {
  if(ks.test(shifted_values.genes[i,], "pweibull", parametri_weibull[i,1], parametri_weibull[i,2])
$statistic < ks.test(M[i,], "pnorm", parametri_normal[i,1], parametri_normal[i,2])$statistic){
    if (ks.test(shifted_values.genes[i,], "pweibull", parametri_weibull[i,1], parametri_weibull[i,2])
$statistic < ks.test(shifted_values.genes[i,], "plnorm", parametri_lognormal[i,1],
parametri_lognormal[i,2])$statistic) {
      best.fit.ks[i,] <- parametri_weibull[i,]
      outdistri[i] <- "weibull"
    }
  } else{
    if (ks.test(M[i,], "pnorm", parametri_normal[i,1], parametri_normal[i,2])$statistic <
ks.test(shifted_values.genes[i,], "plnorm", parametri_lognormal[i,1], parametri_lognormal[i,2])
$statistic) {
      best.fit.ks[i,] <- parametri_normal[i,]
      outdistri[i] <- "normal"
    }
  }
 }
 list(best.fit.ks,outdistri)
}
```

per fittare usando il minimo globale(arrivando ancora più vicini al minimo della popolazione e potendo confrontare i singoli parametri delle distribuzioni in modo più opportuno), il codice è il seguente:

```
best.fit.ks_total <- function(M){
  parametri_weibull <- weibullpar_total(M)
  parametri_normal <- normalpar(M)
  parametri_lognormal <- lognormalpar_total(M)
  best.fit.ks <- parametri_lognormal
  shifted_values.genes <- M - min(M) +0.0001
  outdistri <- rep("lognormal",nrow(parametri_lognormal))
  for (i in 1:nrow(parametri_lognormal)) {
   ks.test.weib <- ks.test(shifted_values.genes[i,], "pweibull", parametri_weibull[i,1],
parametri_weibull[i,2])$statistic
   ks.test.norm <- ks.test(M[i,], "pnorm", parametri_normal[i,1], parametri_normal[i,2])$statistic
   ks.test.lognorm <-  ks.test(shifted_values.genes[i,], "plnorm", parametri_lognormal[i,1],
parametri_lognormal[i,2])$statistic

   if((ks.test.weib < ks.test.norm)
     && (ks.test.weib < ks.test.lognorm)) {
      best.fit.ks[i,] <- parametri_weibull[i,]
      outdistri[i] <- "weibull"
   }
   if((ks.test.lognorm < ks.test.norm)
     && (ks.test.lognorm < ks.test.weib)) {
    best.fit.ks[i,] <- parametri_lognormal[i,]
    outdistri[i] <- "lognormal"
   }
   if((ks.test.norm < ks.test.weib)
     && (ks.test.norm < ks.test.lognorm)) {
    best.fit.ks[i,] <- parametri_normal[i,]
    outdistri[i] <- "normal"
   }
  }
  list(best.fit.ks,outdistri)
 }
```

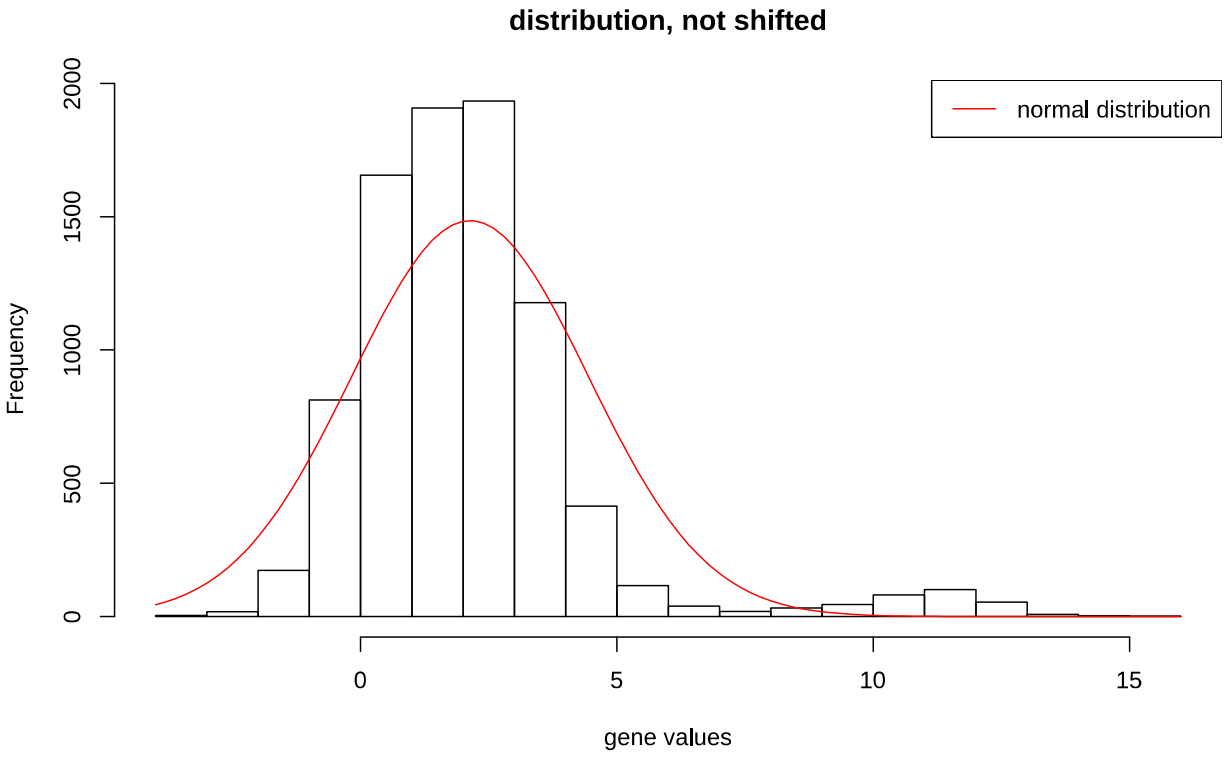
per ogni singolo gene, abbiamo una distribuzione diversa fittata secondo Kolmogorov-Smirnov.

Come esempio, il primo gene si distribuirà con la seguente densità(insieme alla normale fittata):

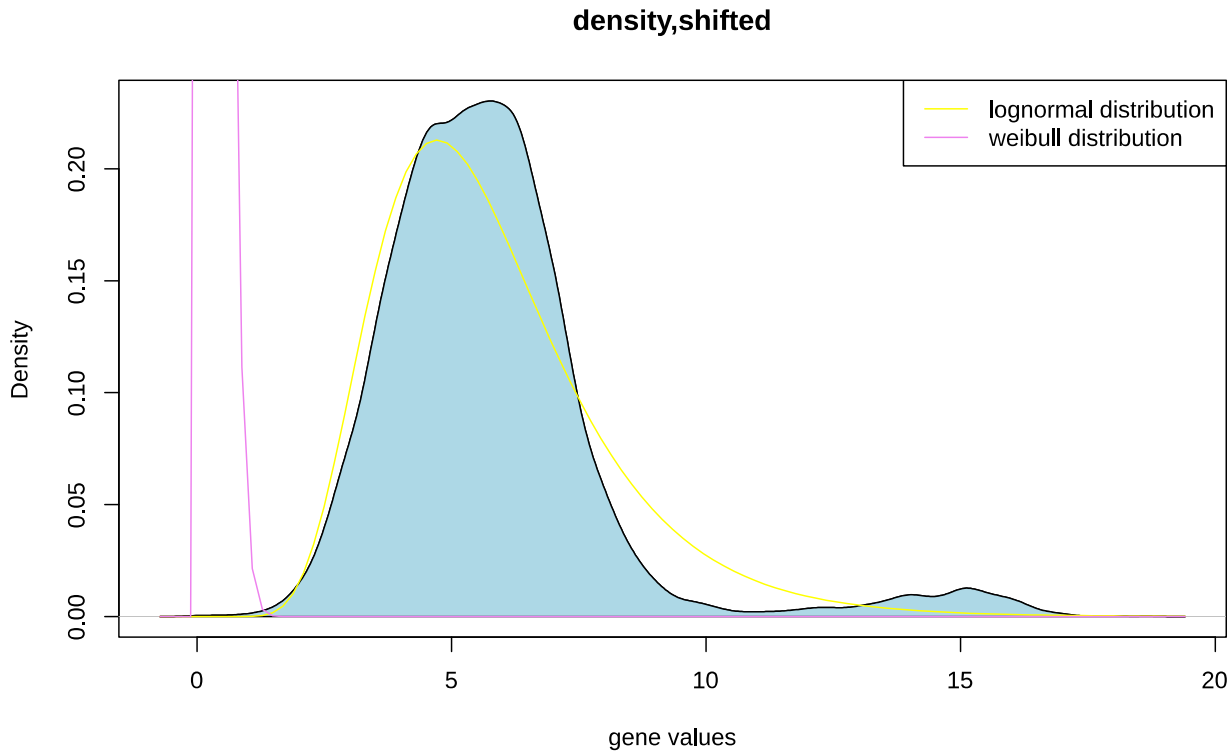


la curva colorata al di sotto è la distribuzione reale del primo gene, mentre l'approssimazione secondo la normale è la curva di colore rosso.

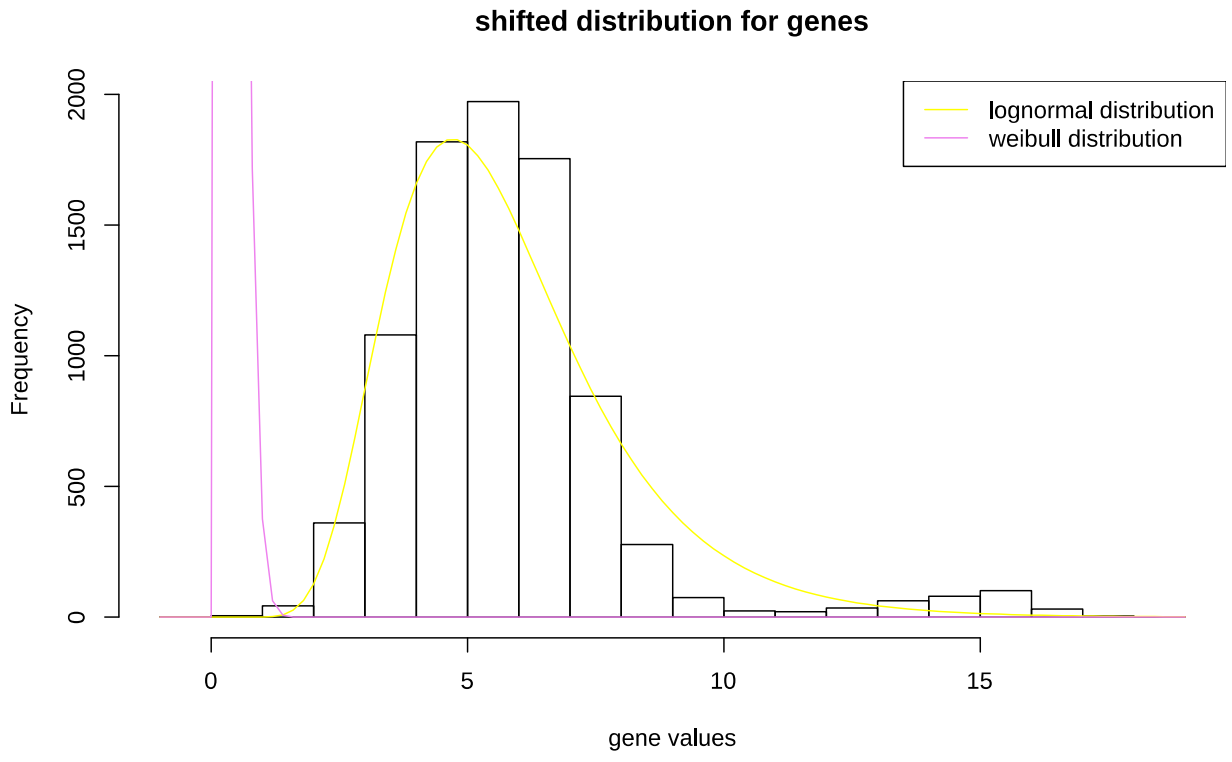
L'istogramma segue lo stesso andamento:



per le distribuzioni di Weibull e Lognormale consideriamo i valori di geni shiftati fino ad essere tutti positivi(traslazione sull'asse x di min(gene values)), la densità di probabilità traslata più i fit di Weibull e Lognormale è la seguente:



la distribuzione è la seguente:



Come si vede dalle densità, la distribuzione che approssima meglio quella reale del primo gene è la Lognormale, infatti la statistica di K-S è la minore tra tutte(il suo p-value è il maggiore anche se non si nota dato il suo valore così piccolo).

Sarebbe comunque giusto specificare che con fit con p-value così piccoli, l'analisi diventà abbastanza approssimativa data la poca goodness dei fit, ma queste considerazioni non sono sempre vere per tutti i geni.

```
> ks.test(training.data.genes[1,]-min(training.data.genes[1,])+0.0000001, "pweibull", parametri_weibull[1,1], parametri_weibull[1,2])

        One-sample Kolmogorov-Smirnov test

data:  training.data.genes[1, ] - min(training.data.genes[1, ]) + 1e-07
D = 0.12563, p-value < 2.2e-16
alternative hypothesis: two-sided

> ks.test(training.data.genes[1,], "pnorm", parametri_normal[1,1], parametri_normal[1,2])

        One-sample Kolmogorov-Smirnov test

data:  training.data.genes[1, ]
D = 0.12299, p-value < 2.2e-16
alternative hypothesis: two-sided

> ks.test(training.data.genes[1,]-min(training.data.genes[1,])+0.0000001, "plnorm", parametri_lognormal[1,1], parametri_lognormal[1,2])

        One-sample Kolmogorov-Smirnov test

data:  training.data.genes[1, ] - min(training.data.genes[1, ]) + 1e-07
D = 0.069118, p-value < 2.2e-16
alternative hypothesis: two-sided
```

CLUSTERING DEI GENI
SUI PARAMETRI DELLE DISTRIBUZIONI

In questo capitolo considererò due tipi di parametri per i singoli geni, con uno fitterò tutti i geni con una distribuzione normale in modo da non modificare le distribuzioni shiftando valori, con l'altro metodo ogni gene avrà una sua distribuzione basata sul fit migliore secondo la migliore misura di goodness di Kolmogorov-Smirnov(le distribuzioni di weibull e lognormale dei geni verranno shiftati del minimo globale), e quindi clusterizzero i geni sulla base della distribuzione a cui appartengono.

Userò principalmente K-means perchè abbiamo troppi punti e pam(partition around medoids) è molto lento nel caso di troppi punti(nella foga del momento ho pure programmato in OpenCL una funzione che calcola le distanze utilizzando la GPU, che alla fine non ho usato dato che il bottleneck era principalmente la funzione pam). Per il calcolo del migliore numero di cluster utilizzerò la **asw**(average silhouette width).

A causa della scelta di Kmeans non posso utilizzare altre metriche di distanza oltre a quella euclidea.

Il codice per clusterizzare utilizzando k-means è il seguente:

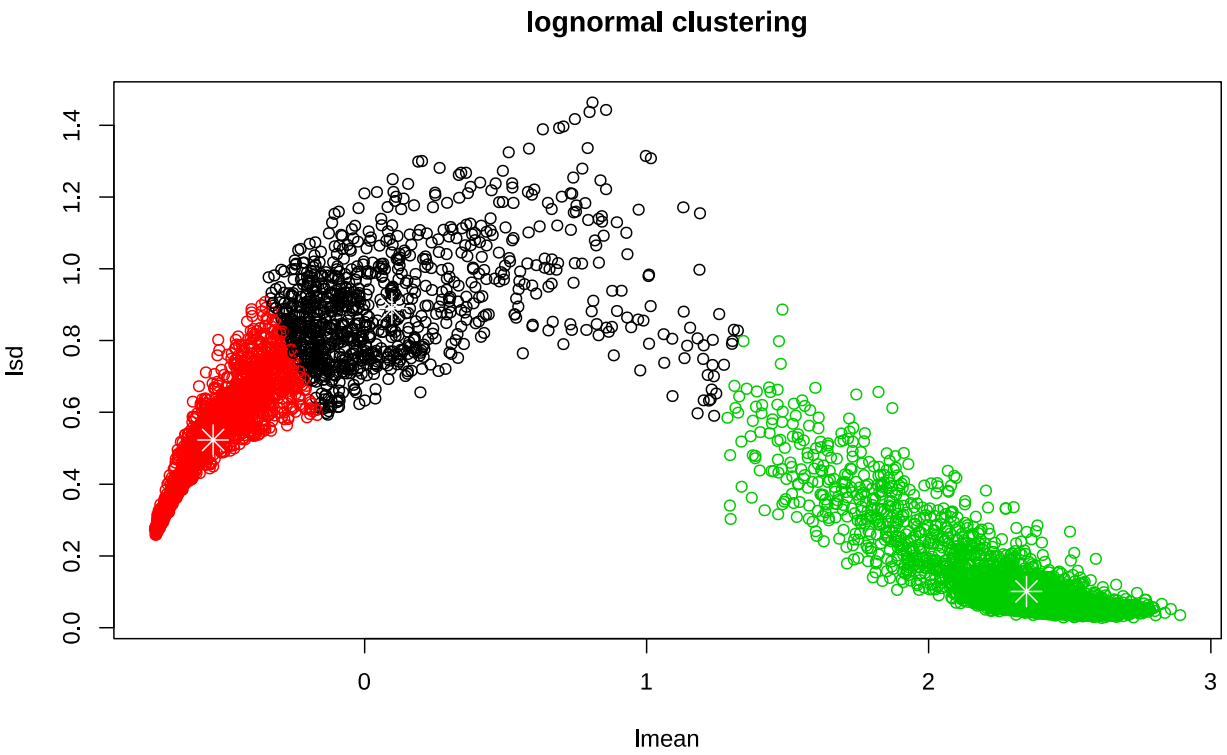
```
optimal_kmeans <- function(M){
 max <- -2
 opt_clust <- -1
 ret_kmeans <- kmeans(M,centers = 3,nstart = 25)
 distances <- parDist(as.matrix(M))
 for (numk in 2:(50)) {
  ret_kmeans <- kmeans(M,centers = numk,nstart = 25)
  silcas <- silhouette(ret_kmeans$cluster,distances)
  if(class(silcas)==”silhouette” ){
   if(summary(silcas)$avg.width < max + 0.1)return(ret_kmeans)
  }
 }
 ret_kmeans
}
```

Personalmente, la funzione silhouette non sembra funzionare bene e continua a darmi valori maggiori nonostante il coefficente di silhouette diminuisca nella realtà.

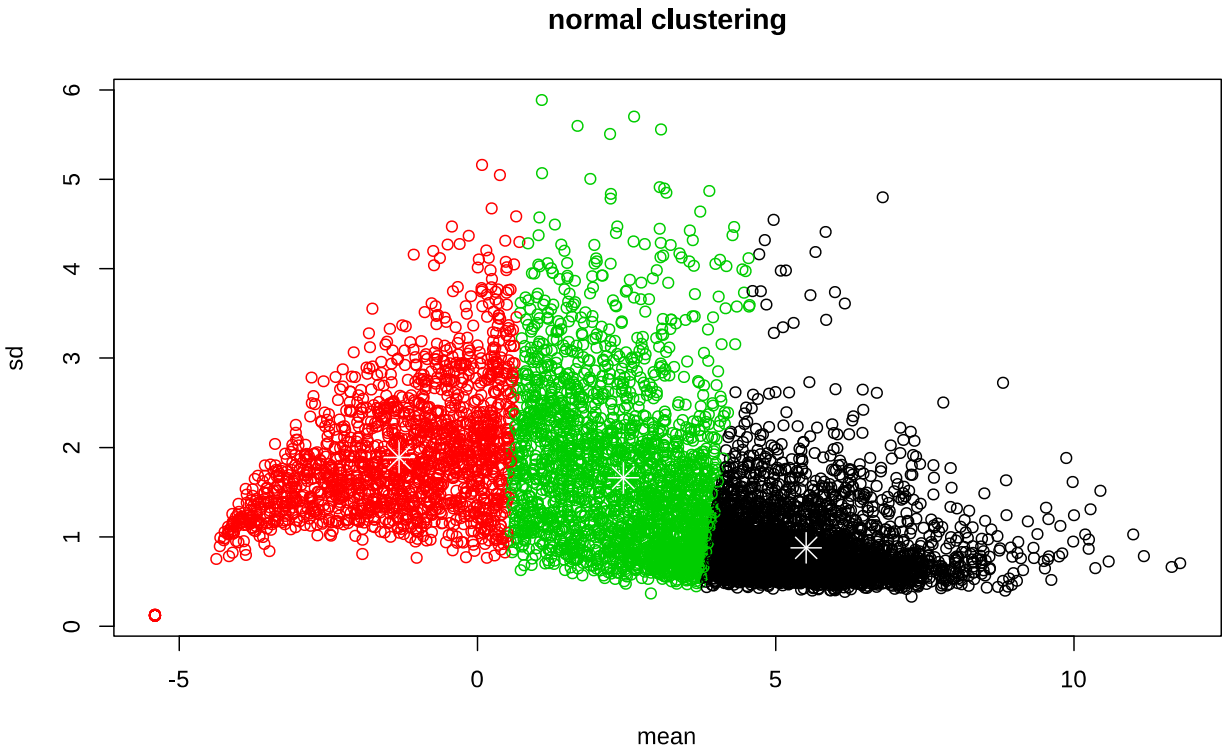
Per ovviare a questo problema mi sono rivolto alla funzione *Optimal_Clusters_KMeans* della libreria **ClusterR**, il codice finale è il seguente:

```
optimal_ext_kmeans <- function(M){
 ottimali <- Optimal_Clusters_KMeans(M,max_clusters = 50,criterion = ”silhouette”)
 kmeans(M,centers = which(ottimali==max(ottimali))+1,nstart = 25)
}
```

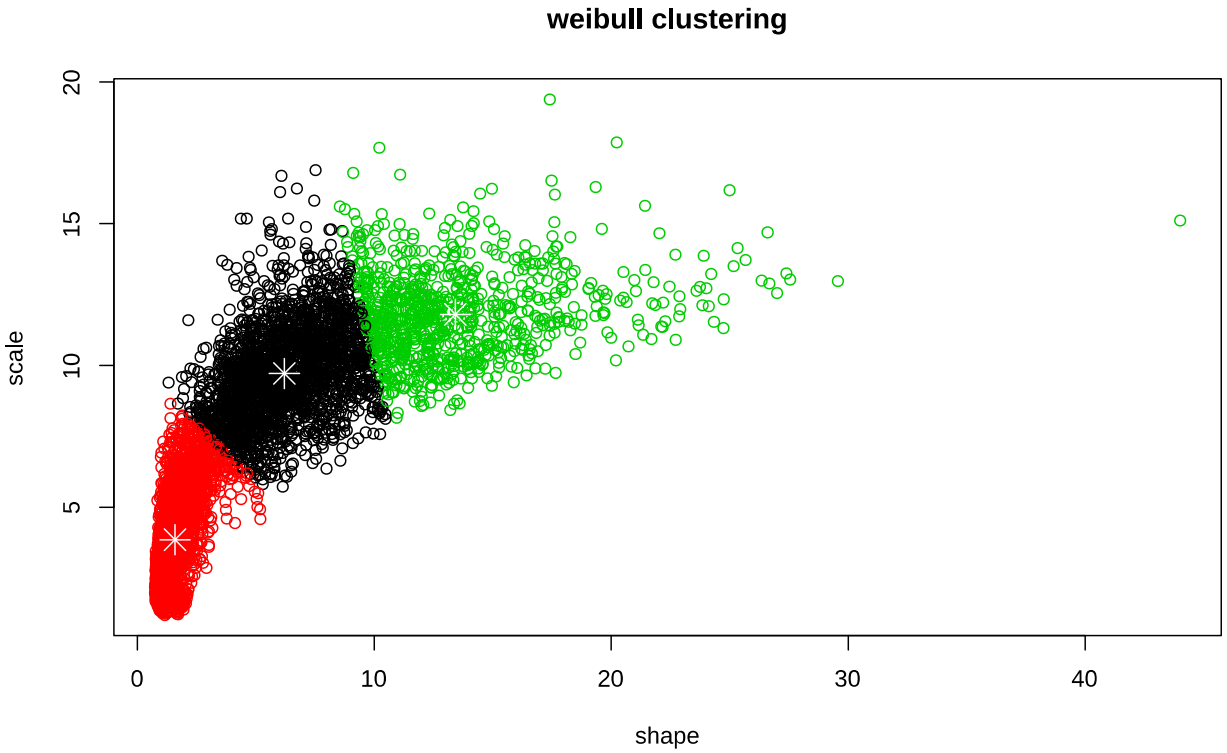
I grafici che rappresentano i cluster sono i seguenti:



lognormal clustering, asse x log-media, asse y log-deviazione-standard



normal clustering,asse x media, asse y deviazione standard

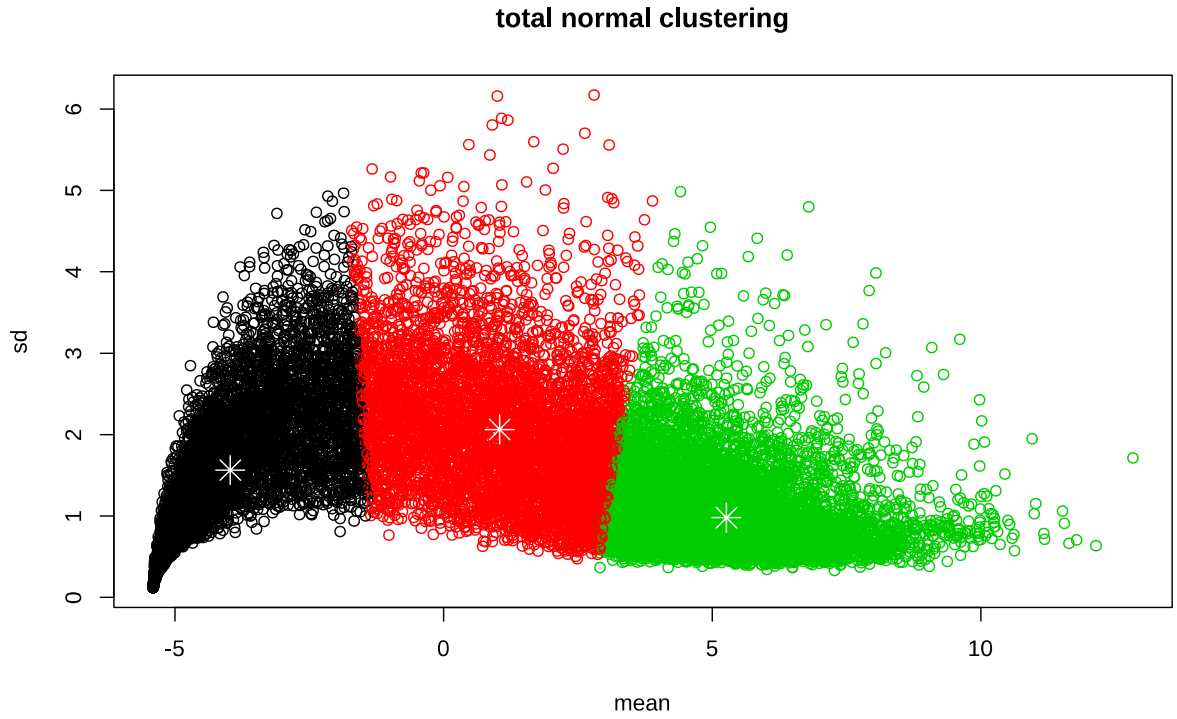


weibull clustering, asse x shape, asse y scale

In realtà l'asw massima si aveva per due cluster, ma mi sembrava poco significativo avere due cluster per così tanti geni(per tre cluster non cambia tanto la situazione).

Come si può vedere, i punti sono troppo vicini tra di loro quindi i cluster saranno pochi.

Il clustering che considera solo la distribuzione normale per ogni gene(senza artefatti da shifting) è la seguente:



TEST DI IPOTESI

Come già detto prima, ci serve un modo per poter clusterizzare i record, e quindi dobbiamo avere delle misure quantitative (potremmo usare altre misure ma dato che stiamo considerando geni è meglio avere valori quantitativi che indichino quanto i geni si avvicinino al valore atteso della distribuzione creata al passo precedente).

Durante questa analisi considereremo due insiemi di dati, uno formato dai p-value per i test unilateri di ogni singolo valore(sarebbe meglio dire che stiamo calcolando le probabilità che $P\{X \geq t\}$ dove t è la variabile aleatoria associata al gene di un individuo), l'altro formato da booleani che rappresentano se la l'ipotesi sia vera o falsa con livello di confidenza di 0.05 nel nostro caso.

Per calcolare il il p-value utilizziamo la funzione di libreria standard **stats** di R pweibull, che calcola la probabilità che $P\{X \geq t\}$ dove t è il valore che assume il record nello specifico gene.

Il codice in R è il seguente per il calcolo del p-value è il seguente(M è la matrice di individui affetti da patologie, par_fit è la lista contenente una matrice Nx2 di N geni, contenente i parametri della distribuzione scelta, ed il vettore di distribuzioni scelte sotto forma di stringhe):

```
funzione_prob_greater_than <- function(M,par_fit){
 ret_mat <- matrix(data = 0,nrow = nrow(M),ncol = ncol(M))
 for (i in 1:nrow(M)) {
  if(fittonewhy[[2]][i]==”normal”){

    ret_mat[i,] <- pnorm(M[i,],mean = par_fit[[1]][i,1],sd = par_fit[[1]][i,2],lower.tail = FALSE)
  }else {
   shift_test <- shifted_weibull(M)
   if(fittonewhy[[2]][i]==”lognormal”){
    ret_mat[i,] <-plnorm(shift_test[i,],meanlog = par_fit[[1]][i,1],sdlog =  par_fit[[1]][i,2],lower.tail =
FALSE)
   } else {
    ret_mat[i,] <- pweibull(shift_test[i,],shape = par_fit[[1]][i,1],scale = par_fit[[1]][i,2],lower.tail = FALSE)
   }
  }
 }
 ret_mat
}
```

Per il calcolo dell'accettazione dell'ipotesi dobbiamo semplicemente confrontare il p-value con il livello di confidenza deciso, il codice è il seguente:

```
funzione_ipotesi <- function(M,par_fit,significance){
 pval_mat <- funzione_prob_greater_than(M,par_fit)
 ret_mat <- matrix(FALSE,nrow = nrow(pval_mat),ncol = ncol(pval_mat))
 for (i in 1:nrow(pval_mat)) {
  for (j in 1:ncol(pval_mat)) {
    ret_mat[i,j] = pval_mat[i,j]>significance
  }
 }
 ret_mat
}
```

oppure per un codice più ottimizzato(che è stranamente più lento) possiamo vettorizzare:

```
funzione_ipotesi_opt <- function(M,par_fit,significance){
  greaterthanfunc <- function(x,y,mat,alpha){
    mat[x,y] <- (mat[x,y]>alpha)
  }
  greaterthanfunc_vect <- Vectorize(greaterthanfunc,vectorize.args = c("x","y"))
  outer(1:nrow(M),1:ncol(M),greaterthanfunc_vect,funzione_prob_greater_than(M,par_fit),significance)
}
```

CLUSTERING

Il cuore di questa analisi è raggruppare individui differenti con corredi genetici simili, in modo da scoprire se condividono qualche patologia.

Abbiamo diversi modi per poter clusterizzare I nostri dati, ma prima dobbiamo definire delle misure di distanza che ci permettano di misurare in modo concreto e logico la distanza tra due individui.

Durante questa analisi, verranno usate due tipi di misure in base ai dati in nostro possesso(che siano i p-value associati al corredo genetico di ogni individuo, o che siano I booleani associati all'ipotesi descritta in precedenza).

Per i p-value associati ad ogni individuo per il suo corredo genetico, possiamo usare una qualsiasi misura, ma una scelta logica sarebbe una metrica che tenga conto della piccola distanza tra i punti(i p-value sono tutti compresi tra 0 e 1) e della grande quantità di dimensioni di ogni corredo genetico preso in esame, quindi il coseno di dis/similitudine sarebbe una scelta appropriata .

La similarità del coseno tra due vettori(record) viene definita come segue

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

quella che si usa nel caso delle distanze è il complemento della similarità, cioè 1 – *similarity.*

Per avere una misura di vicinanza dai record della matrice di booleani di ipotesi, possiamo usare la Jaccard distance, definita tramite la Jaccard similarity

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

per definire una distanza, allo stesso modo della cosine distance, si calcola il complementare della jaccard similarity.

Queste due distanze non ci sono nella libreria standard **stats** di R, per calcolare queste distanze useremo la funzione *dist* sovrascritta da quella nel pacchetto **proxy**, dove possiamo trovare le misure di distanza che ci interessano.

```
distance_vec <- dist((funzione_prob_greater_than(esem.matr)),method = "cosine",by_rows =TRUE)

distance_vec.ipotesi <- dist(t(funzione_ipotesi(esem.matr,0.05)),method = "jaccard",by_rows =TRUE)
```

Per clusterizzare si possono usare vari approcci, quelli che considererò durante questo studio sono principalmente k-means e clustering gerarchico.

Usero una versione adattiva e più robusta di k-means in cui viene considerata la dissimilarity matrix (solitamente non vengono considerate altre distanze oltre a quella euclidea per il fatto che, come

specificato dal nome, deve essere possibile trovare il medoide-clusteroide in qualche modo, e la distanza euclidea è la più semplice da utilizzare anche grazie alle sue proprietà).

Le funzioni di R che verranno usate saranno *pam e pamk* (partition around medoids) della libreria **fpc** che prendono la matrice di dissimilarità(o alternativamente direttamente le osservazioni), la versione *pamk* calcola direttamente anche la silhouette e sceglie il numero di cluster che massimizza la ampiezza media di silhouette(dettagli sulla silhouette nel prossimo capitolo) .

Il codice in R per clusterizzare i due insiemi di dati(p-value e ipotesi) utilizzando k-means è il seguente:

```
cluster_pvalue_cos <- function(M){
 distance_vec <- dist(t(funzione_prob_greater_than(M)),method = "cosine")
 pamk(distance_vec,diss = TRUE,krange = 1:(ncol(M)-1),criterion = "asw",usepam = TRUE)
}

cluster_ipotesi_jac <- function(M){
 distance_vec <- dist(t(funzione_ipotesi_opt(M,0.05)),method = "jaccard")
 pamk(distance_vec,diss = TRUE,krange = 1:(ncol(M)-1),criterion = "asw",usepam = TRUE)
}
```

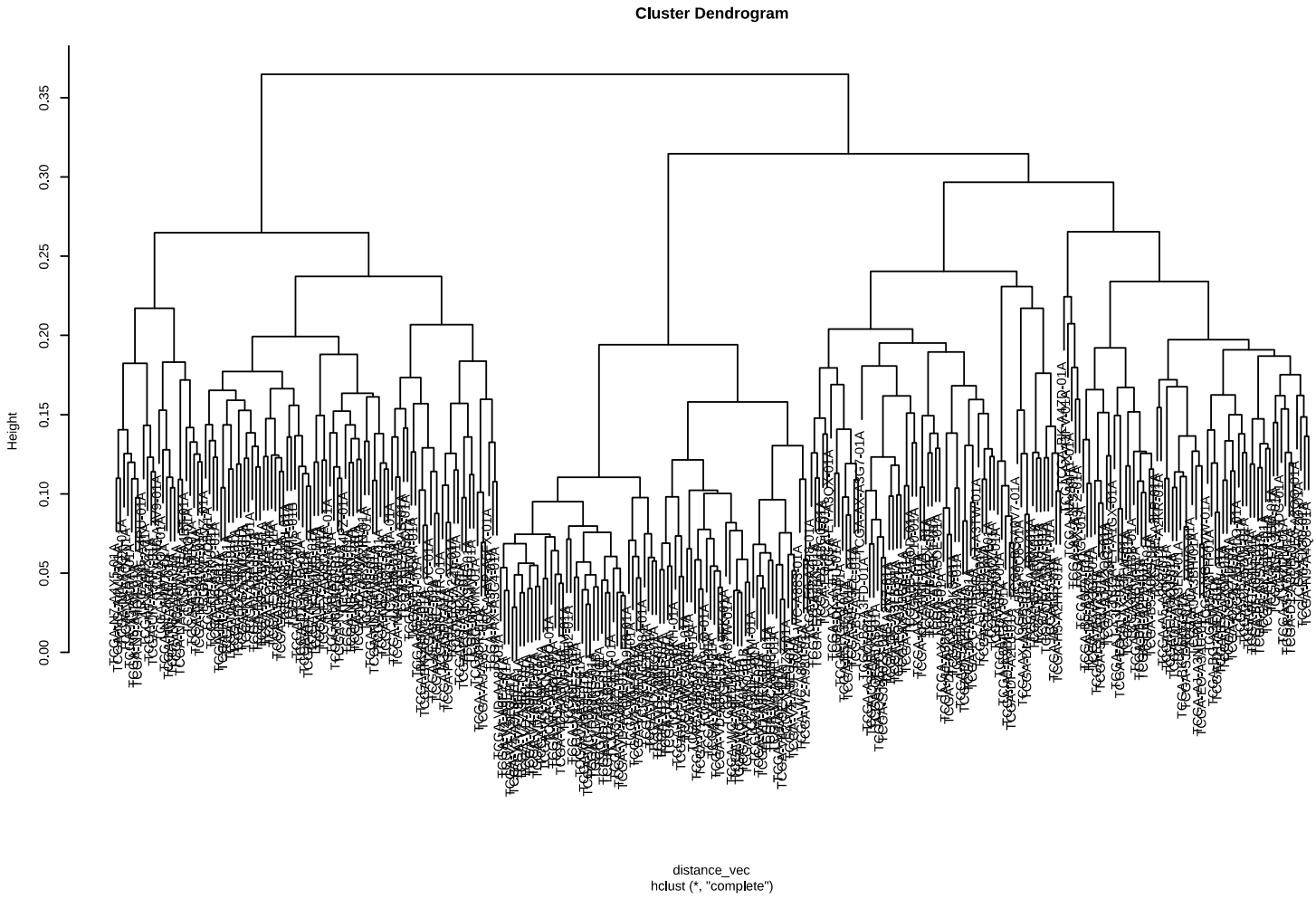
Per il clustering gerarchico verra usata la funzione della libreria standard **stats** *hclust*, che restituisce il dendrogramma totale(singolo cluster), vogliamo trovare il numero ottimale di cluster, useremo quindi delle misure che possano valutare la qualità del nostro modello di clusterizzazione in modo da riuscire a trovare il numero giusto di cluster, queste misure verranno viste nel prossimo capitolo

Il codice in R per clusterizzare utilizzando hclust è il seguente:

```
dendro.cluster.pvalue_func <- function(M,par_fit){
  M <- out_no_na_col(M)
  distance_vec <- dist((funzione_prob_greater_than(M,par_fit)),method = "cosine",by_rows = FALSE)
  hiera_cluster.pvalue <- hclust(distance_vec)
 }

hcluster_pvalue_cos <- function(M,par_fit){
 distance_vec <- dist((funzione_prob_greater_than(M,par_fit)),method = "cosine",by_rows = FALSE)
 hiera <- hclust(distance_vec)
 opt_clust <- optimal_numclast(hiera,distance_vec)
 list(cutree(hiera,opt_clust),opt_clust)

}


hcluster_ipotesi_jac <- function(M,par_fit){
 distance_vec <- dist((funzione_ipotesi(M,par_fit,0.05)),method = "jaccard",by_rows = FALSE)
 hiera <- hclust(distance_vec)
 opt_clust <- optimal_numclast(hiera,distance_vec)
 list(cutree(hiera,opt_clust),opt_clust)
}
```

dove la funzione *optimal_numclast* calcola il numero che massimizza l'ampiezza media della silhouette(la vedremo in dettaglio nel prossimo capitolo).

Il dendrogramma di un sottocampione si presenterà nel seguente modo.



MISURE DI QUALITÀ DEL CLUSTERING

Molte misure possono essere usate per valutare la qualità dei cluster in qui sono stati divisi i dati in nostro possesso, per esempio la «Within cluster sum of squares by cluster» oppure la compactness, che misura la compattezza dei cluster ottenuti (compactness più alta indica un miglior clustering), ma ci affideremo ad una misura che viene usata molto anche per la sua robustezza e per i risultati che si ottengono, questa misura è il coefficiente di silhouette.

Il coefficient di silhouette si utilizza per quantificare la qualità dei cluster ottenuti.

a) Per ogni osservazione *i*, calcola la distanza media tra *i* e tutti gli altri

punti dello stesso cluster a cui *i* appartiene. Sia D_i tale distanza;

b) Per ogni osservazione *i*, calcola la distanza tra *i* e il cluster più vicino

a *i* (escluso ovviamente quello a cui i appartiene). Sia C_i tale distanza;

c) Il coefficiente di silhouette S_i è dato da:

$$S_i = \frac{C_i - D_i}{\max(C_i, D_i)}$$

• $S_i > 0$ indica che l'osservazione *i* è ben clusterizzata. Più è vicino a 1

e meglio è clusterizzata.

• $S_i < 0$ indica che l'osservazione è stata piazzata in un cluster

sbagliato.

• $S_i = 0$ indica che l'osservazione è a metà tra due cluster.

La misura che verrà usata per calcolare il numero di cluster ottimale sarà la media tra tutti i coefficenti di silhouette, chiamata **ampiezza media di silhouette**(average silhouette width, abbreviato **asw**).

La funzione pamk calcola direttamente il numero ottimale di cluster che massimizza **asw**.

Per il clustering gerarchico dobbiamo cercare il numero ottimale utilizzando **asw** per arrivare alla massima qualità ottenibile.

Come descritto nel capitolo precedente, per il clustering gerarchico cerchiamo il numero di cluster ottimale, per trovare questo numero possiamo usare due approcci diversi, o una strategia lineare dove confrontiamo tutti i risultati per $k \in [2,n]$, oppure una ricerca binaria.

Il codice per trovare il numero di cluster ottimale massimizzando **asw** è il seguente(versione lineare e binaria):

```
optimal_numclast <- function(hierarch,distances){
 max <- -2
 opt_clust <- -1
 for (numk in 2:(ncol(distances)-1)) {
  silcas <- silhouette(cutree(hierarch,numk),distances)
  if(class(silcas)==”silhouette” ){
   if(summary(silcas)$avg.width > max)
    opt_clust <- numk
   else return opt_clust
  }
 }
 opt_clust
}

optimal_numclast_binary <- function(hierarch,distances){
 l <- 2
 m <- 0
 r <- ncol(distances)-1
 max <- -2
 while ( l<=r ){
  m <- l + ((r - l)* 2)
  silcas <- silhouette(cutree(hierarch,m),distances)
  if (class(silcas)==”silhouette” && summary(silcas)$avg.width > max ){
   max <- summary(silcas)$avg.width
   l = m +1;
  }
  else
   r = m -1;
 }
 m
}
```

PATOLOGIE FREQUENTI NEI CLUSTER

L'obiettivo di questo studio è trovare delle relazioni tra individui con corredi genetici simili e vedere se condividono qualche patologia particolare.

A partire dal vettore di individui con i loro corredi genetici bisogna avere a disposizione anche un altro insieme di dati che rappresenta sempre gli stessi individui per cui si sono calcolate le matrici descritte nei capitoli precedenti, ma al posto dei geni abbiamo attributi che rappresentano patologie possedute dagli individui.

Dopo aver calcolato tutte le matrici che ci servono ed avere clusterizzato in modo ottimale, otteniamo il vettore di appartenenza ai cluster che ci dice a quale cluster appartiene un individuo.

Per trovare gli attributi più frequenti per ogni cluster dobbiamo semplicemente isolare gli individui appartenenti ai singoli cluster e vedere quale è la patologia con più frequenza(la moda), se ci sono più mode prendiamo la prima(scelta soggettiva, possiamo restituire anche un vettore di patologie).

Il codice per trovare la moda(la prima) delle patologie in ogni cluster insieme alla frequenza è il seguente:

```
attr_vect.cluster <- function(patologie_vect,cluster_vect,numk){
 vectret <- c(rep("",numk))
 vectnum <- rep(0,numk)
 for (i in 1:numk) {
  tt <- table(patologie_vect[cluster_vect== i])
  vectret[i] <- names(tt[tt==max(tt)])[1]
  vectnum[i] <- max(tt)
 }
 list(vectret,vectnum)
}
```

Bisogna comunque dire che la quantità di cluster che massimizza la *asw* è quasi sempre molto piccolo, quindi clusterizzare per patologie potrebbe risultare una analisi non troppo significativa data la numerosità dei cluster ottenuti.