Facultad de Matemáticas

Departamento de Estadística e Investigación Operativa

Trabajo Fin de Grado en Estadística

TÉCNICAS DE ANÁLISIS DE EXPRESIÓN DIFERENCIAL BASADAS EN CONTEOS PARA EL ESTUDIO DE DATOS DE RNA-SEQ USANDO R Y BIOCONDUCTOR

Sara Carrasco Carrasco

2 de diciembre de 2015

Dirigido por:
Dra. Inmaculada Barranco Chamorro
Dr. Pedro Luis Luque Calvo

Índice general

Capítulo 1

Introducción

La importancia que el análisis estadístico de datos genéticos ha adquirido actualmente se debe en gran parte a sus múltiples aplicaciones médicas, como por ejemplo, el estudio de la eficacia que cierto tratamiento tiene sobre una enfermedad. Durante este capítulo, se desarrollará el método de secuenciación de ARN denominado RNA Sequencing (*RNA-Seq*), una de las tecnologías que utiliza la ultrasecuenciación para detectar y cuantificar la cantidad de ADN de un genoma. Además se explicará posibles tratamientos a aplicar a los datos para poder realizar el análisis estadístico con los datos de *RNA-Seq*, en concreto las técnicas de normalización de datos.
Se puede obtener más información sobre éste método en Gonzalez [1].

1.1. Datos de *RNA-Seq*

El término ciencias ómicas recoge disciplinas que se basan en el análisis de un gran volumen de datos. En el momento en el que estas ciencias logran poner a disposición las secuencias de los genomas, se empiezan a considerar nuevos tipos de estudios y surgen preguntas tales como la relación existente entre el perfil de expresión genética y la eficacia de ciertos tratamientos.

La técnica de *microarrays* fue una de las primeras que trataron de estudiar el nivel de expresión de todos los genes de forma simultánea. Un DNA *microarrays* consiste en una superficie sólida a la que se le une una colección de fragmentos de ADN y en la que se pueden observar a través de fluorescencia, el nivel de expresión de cada gen, por lo que se tendría unos datos de tipo continuo.

Un análisis de este tipo partiría de la información recogida en una placa como la que se muestra a continuación.

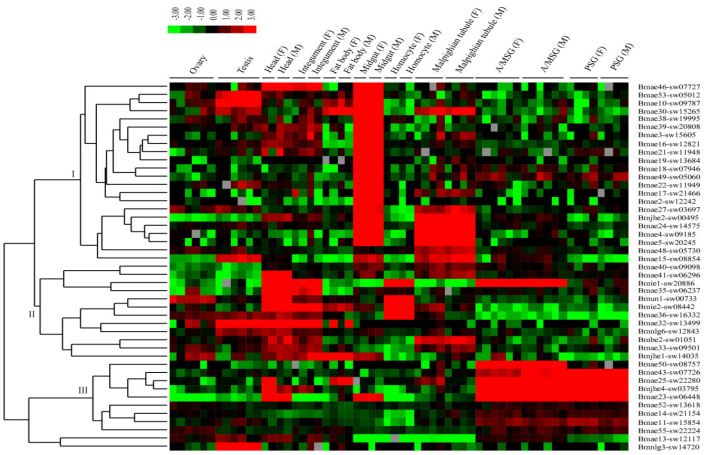


Figura 1.1: DNA microarray

Pero esta técnica mostraba limitaciones, como que en determinados momentos los niveles de fluorescencia se saturan, y como, lógicamente, las tecnologías han ido evolucionando surgieron las denominadas HTS (*high-throughput sequencing* - técnicas de secuenciación de última generación). Llegando así, a través del perfeccionamiento de las técnicas, a las llamadas *RNA-Seq*, que se pueden definir como una aproximación para realizar el análisis de perfiles de expresión utilizando tecnología HTS.

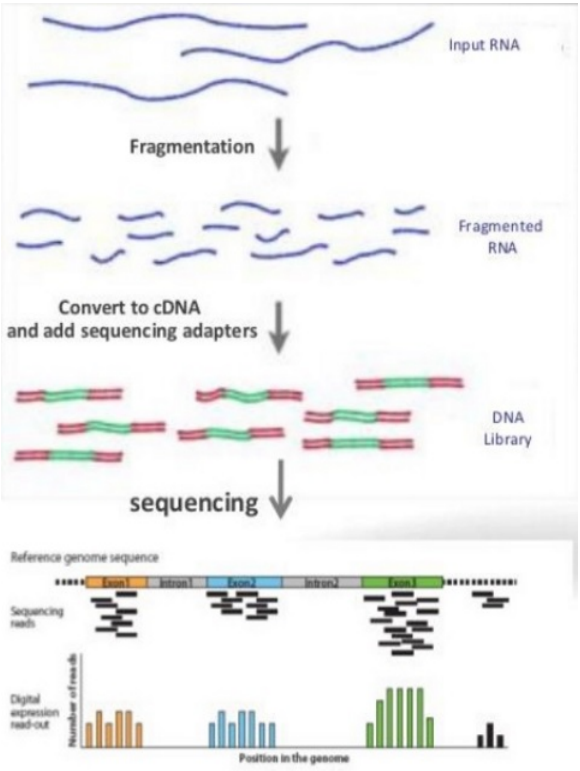


Figura 1.2: Flujo de trabajo de RNA-Seq

Como se muestra en la Figura 1.2, el primer paso que sigue esta técnica es la fragmentación del ADN en pequeños trozos, llamados *reads* o lecturas, luego se secuencia cada *read* y finalmente las alinea frente a un genoma de referencia estándar. Para la expresión genética, se alinean frente a genes, de forma que un gen se dirá que posee una expresión mayor que otro si el número de *reads* alineados contra ese gen es mayor.
De esta alineación obtenemos unos datos de conteo, los cuales se resumirán en tablas para su estudio estadístico. Las tablas de conteos que se obtienen al aplicar RNA-Seq tienen la siguiente forma:

	$I_1^A, ..., I_n^A$	$I_1^B, ..., I_m^B$
Gen 1		
.		
.		
.		
Gen G		

Cuadro 1.1: Formato tabla de conteos.

En las columnas se mostrarán dos grupos, los cuales serán el objeto del estudio estadístico posterior, cada uno de ellos con un número determinado de individuos. Un ejemplo podría ser: Grupo A - individuos tratados con un cierto medicamento, Grupo B - individuos no tratados o grupo control. Por otro lado, en las filas estarán cada uno de los genes. Así cada celda de la tabla mostrará los conteos que el individuo de la columna en la que se encuentre ha recibido del gen correspondiente.

Una vez recogidos los datos y resumidos en tablas de conteos, se pueden realizar diversos estudios. El objetivo de nuestro estudio es identificar los genes que son diferencialmente expresados (DE). Esto lo haremos mediante una comparación de medias. Sea X_g la variable aleatoria que nos muestra los conteos del gen g, y sean μ_A^g y μ_B^g las medias correspondientes de esta variable aleatoria a los grupos A y B, respectivamente, se trataría de un problema de comparación de medias de dos grupos independientes.
Los modelos estadísticos útiles para este tipo de análisis son los modelos para datos de conteos, que se explicarán en los capítulos 2, 3 y 4.

1.2. Consideraciones previas al análisis de datos

Antes de analizar los datos es importante tener en cuenta algunos puntos sobre estos:

- En primer lugar los datos obtenidos en un experimento de RNA-Seq, proporcionan cierto número de lecturas, cuya agrupación proporciona las llamadas librerías de secuenciación. El número total de lecturas mapeadas en el genoma determina la profundidad de secuenciación del ensayo, determinando a su vez el tamaño de la librería.

- A la longitud de cada gen se le debe prestar especial atención a la hora de analizar los datos, pues los genes con longitudes reducidas pueden mostrar menos lecturas que los genes con una longitud mayor, por lo que interesaría poner en una misma escala las longitudes de cada gen.

- Y por último, es necesario conocer el tipo de réplica con la que se trabaja. Existen dos tipos de réplicas, por un lado están las biológicas, donde en general, cada individuo perteneciente al estudio representaría una muestra, y por otro lado las réplicas técnicas, que son muestras tomadas de un mismo individuo, por lo que finalmente se trabajará con la media de todas ellas.
  En este último tipo de réplica no es posible extraer conclusiones generales, sino que solo son válidas para muestras que puedan ser directamente comparables.

1.3. Análisis de datos

Como ya se ha explicado en la Sección 1.1, los datos resumidos que obtenemos al aplicar RNA-Seq son datos de conteos, por lo que al contrario que en la técnica de *microarrays* tenemos datos discretos. Para poder trabajar con ellos existen dos posibilidades:

1. **Transformar los datos discretos en datos continuos.** Así se podría realizar el estudio aplicando técnicas de *microarrays*.
   Para proceder de esta forma, primero se tiene que tomar logaritmos de los datos y luego aplicar una transformación que estabilice la varianza. Aunque tendríamos el problema de qué hacer con los conteos que toman valor cero en los datos originales, pues no se puede calcular el logaritmo de ellos.
   En caso de realizar esta transformación, se actuaría como si tuvieramos datos procedentes de *microarrays* y se seguiría los procedimientos disponibles para ellos.

2. **Utilizar modelos estadísticos específicos para datos de conteo.** Esta es la mejor opción, pues se trabajaría con los propios datos, sin realizar transformaciones, ya que se utilizará un modelo específico para los datos. Los modelos que se utilizarían son los Modelos Lineales Generalizados que se explicarán en el siguiente capítulo.

Aunque, debido a la opción elegida para trabajar con los datos, estos no vayan a ser transformados, si deberán de ser normalizados.

1.4. Normalización

Antes de comenzar con el análisis, se deben de normalizar los datos, sin embargo en este contexto *normalizar* tiene un significado distinto al que es usual en estadística. En RNA-Seq con la normalización se busca minimizar el ruido técnico introducido en los datos durante el proceso de secuenciación con el fin de volverlos

comparables entre sí, no se pretende transformarlos para que sigan una distribución normal, que es lo que se suele entender como *normalización* en términos estadísticos.

Al normalizar buscamos dos objetivos, el primero es poner en una misma escala todas las muestras o individuos correspondientes a cada tratamiento para evitar falsos positivos, ya que una muestra o librería con mayor profundidad de secuenciación tiene más probabilidad de tener genes DE respecto a otra, sin deberse estas diferencias a la condición bajo estudio.
El segundo objetivo es menos obvio, se trata de eliminar las variaciones biológicas entre las muestras. La tecnología RNA-Seq proporciona una medida de la abundancia relativa de cada gen en cada muestra de RNA, pero no hace un "cálculo general" de esta abundancia. El problema viene cuando un pequeño número de genes pueden consumir una proporción considerable del total, es decir, cuando unos genes se expresan mucho para algunas muestras y apenas nada para otras, o cuando existen unos genes que prácticamente son la mayor proporción de la expresión de estos, pues si se hiciera un recuento global de los conteos realmente estos abarcarían mucho en esa proporción y podría llegar a hacernos una idea equivocada de la expresión de los genes en general, esto se debe a la longitud de cada gen en cuestión. Un gen es una secuencia de nucleótidos y a cada nucleótido le corresponde una base, el número de bases que posea un gen, determina la longitud del mismo. Un gen con una longitud menor puede mostrar un número menor de lecturas que otro con mayor longitud. Por lo que los genes no están en una misma escala y puede llegar a ocurrir lo citado anteriormente.

Existen diversos métodos para llevar a cabo la normalización de estos datos:

- Uno de los más utilizados, aunque no es muy sofisticado es el de dividir los conteos que tenemos para cada gen en una muestra por el número total de lecturas en dicha muestra.

- Otro método, es el denominado **RPKM** (Reads Per Kilobase of transcript and Million mapped reads) lo que se traduciría como lecturas esperadas por kilobase de tránscrito y millón de lecturas mapeadas.
  Este método normaliza por la longitud de las regiones en cuestión, por ejemplo, si queremos normalizar por la longitud de los genes sería:

$$RPKM = \frac{\frac{número\ de\ lecturas\ en\ la\ región}{amplitud\ de\ la\ región\ \times 10^3}}{número\ total\ de\ lecturas\ \times\ 10^6}$$

  Se puede encontrar más información sobre este método en Mortazavi et al. [2].

- Otro es el método **TMM** (Trimmed Mean of M-values), es decir, media truncada de M - valores, que fue propuesto por Robinson and Oshlack [3].
  La base de este método sería la siguiente:
  Sean Y_{gk} y μ_{gk} el número de conteos observado y el nivel de expresión para el gen g en la muestra k, respectivamente. El nivel de expresión será desconocido y vendrá dado en términos del número de transcripciones. Sean L_g y N_k la

longitud del gen g, es decir, el número de bases que éste adquiere y el número total de lecturas en la muestra k.
El valor esperado de Y_{gk} es:

$$E[Y_{gk}] = \frac{\mu_{gk} L_g}{S_k} N_k$$

Siendo

$$S_k = \sum_{g=1}^{G} \mu_{gk} L_g$$

Donde S_k representa la salida de RNA total de la muestra. Este método, aunque es bastante recomendable, tiene la dificultad de que el valor S_k es desconocido y se estima apartir de la muestra, por lo que puede variar drásticamente de una muestra a otra.
Basándose en este método se calculan medias truncadas de medidas de la abundancia relativa de cada gen entre dos muestras.

Dependiendo del sofware utilizado será necesario realizar la normalización antes de analizar los datos, o la realizará internamente el programa. En capítulos posteriores se estudiarán varios paquetes del sofware R y Bioconductor. En concreto en **DESeq** y **DESeq2** utilizan el primer método anteriormente citado y en **edgeR** utilizan el método TMM.

Una vez que ya está el proceso de RNA-Seq explicado y el tratamiento que debe de recibir los datos obtenidos por éste para poder proceder al análisis, el siguiente paso es ajustar los datos a un modelo, que como veremos en el siguiente capítulo, los Modelos Lineales Generalizados son los que mejores se ajustan a los datos de conteo.

Capítulo 2

Modelos Lineales Generalizados.

En el capítulo anterior se propuso como la opción más adecuada para realizar un estudio estadístico de datos procedentes de *RNA-Seq* utilizar modelos estadísticos específicos para datos de conteo. Vamos a estudiar los Modelos Lineales Generalizados (MLGs). Los MLGs son una extensión de los modelos lineales clásicos los cuales tratan de explicar la variable objetivo como una combinación lineal de una función de un conjunto de variables explicativas. Podemos encontrar un desarrollo más amplio de los Modelos Lineales Generalizados en Hardin and Hilbe [4] y Hilbe [5].

2.1. Introducción.

El modelo lineal clásico expresa la esperanza condicionada de la variable objetivo como combinación lineal de las variables explicativas. El modelo muestral en este caso, teniendo y como la variable respuesta o variable objetivo, y $\underline{x}$ como el vector con las variables explicativas sería:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p + \epsilon, \quad \epsilon \sim N(0, \sigma^2) \ independientes$$

Sin embargo, mediante la reestructuración de la relación entre el predictor lineal y el ajuste, se pueden linealizar relaciones que inicialmente parecen ser no lineales y por tanto no sería válido este modelo lineal clásico, pues esta modelización surge de la linealidad de la esperanza condicionada en la distribución normal. Pero existe una generalización basada en de plantear una hipótesis distribucional y una estructural, que nos llevará a los MLGs.

1. **Hipótesis distribucional.** Nos indica que la variable objetivo condicionada a las variables explicativas son independientes con una distribución perteneciente a la familia exponencial simple. La función de densidad de esta familia es:

$$f(y; \theta, \phi) = exp\left\{\frac{y\theta - b(\theta)}{\phi} + c(y, \phi)\right\} \tag{2.1}$$

Siendo θ y ϕ los parámetros natural y de escala o dispersión, respectivamente, *b(·)* es la función cumulante, *c(·)* una función específica para cada distribución de la familia y *f(·)* la función de densidad en el caso continuo, y la función de probabilidad en el caso discreto.

2. **Hipótesis estructural.** Esta hipótesis recoge la relación que se supone que existe entre la esperanza, μ, de la variable objetivo condicionada, $Y|_{\underline{X}=\underline{x}}$, y un predictor lineal $\underline{z}^t\underline{\beta}$, donde $\underline{z}^t = (1, x_1, ..., x_p)$ y $\underline{\beta} = \beta_0 + \beta_1 + ... + \beta_p$, a través de la función:

$$\mu = h(\underline{z}^t\underline{\beta}) \quad \text{ó} \quad \underline{z}^t\underline{\beta} = g(\mu)$$

Siendo $\mathbf{h(\cdot)}$ la función respuesta; $\mathbf{g(\cdot)}$ la función link, que es la inversa de h(·); $\underline{\beta}$ el vector de parámetros desconocidos del modelo y $\underline{z}$ el vector de diseño, que muestra la combinación de las variables explicativas del modelo.

Por tanto, para tener totalmente especificado el modelo, necesitamos conocer, la distribución de la familia exponencial adecuada, la función vínculo y el vector de diseño. Respecto a la fucnión vínculo, para cada distribución de la familia exponencial existe una determinada, que relaciona el parámetro natural, θ, de la distribución correspondiente con el predictor lineal del modelo, $\underline{z}_i^t\underline{\beta}$.

$$\theta = \theta(\mu) = \underline{z}^t\underline{\beta}\ ; \quad g(\mu) = \theta(\mu) \tag{2.2}$$

Por ejemplo, para la regresión de Poisson, un caso particular del Modelo Lineal Generalizado y de la familia exponencial de distribuciones, la función que relaciona ambas características, es decir, $\theta(\mu)$, es $ln(\mu)$, por lo que tendríamos:

$$\theta(\mu) = ln(\mu) \Longrightarrow ln\mu = \underline{z}^t\underline{\beta} = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p$$

De esta forma, teniendo unos datos de conteos, con determinar que distribución ajustar a estos según las características que estos presenten, ya se tendrían los tres requisitos nombrados anteriormente y necesarios para tener totalmente especificado el modelo a utilizar.

2.2. Estimación de máxima verosimilitud

La inferencia en el modelo anteriormente explicado se basa en la función de verosimilitud. Se trata de maximizar la verosimilitud para obtener un estimador del vector de parámetros desconocidos $\underline{\beta}$ del modelo. Para ello el primer paso es suponer el parámetro ϕ conocido, y como en la verosimilitud aparecerá, asumiremos que $\phi = 1$, pues no se perderá generalidad con esta suposición, tras calcular los estimadores buscados se podrá estimar éste a partir del método de los momentos. También supondremos que la matriz de diseño, compuesta por los vectores de diseño de cada observación, tiene rango total.

Con una muestra de n individuos, considerando que cada familia distribucional tiene unos valores diferentes para determinados parámetros, la contribución de la i-ésima observación en el logaritmo de la función de verosimilitud (log-verosimilitud), es:

$$l_i(\theta_i) = ln\ f(y_i;\theta_i,\phi) = \frac{y_i\theta_i - b(\theta_i)}{\phi} + c(y_i,\phi)$$

En caso de utilizar la función vínculo natural, dada en (2.2), tendríamos la siguiente expresión para $l_i(\theta_i)$ en términos del predictor lineal del modelo.

$$l_i(\theta_i) = \frac{y_i\theta_i - b(\theta_i)}{\phi} + c(y_i,\phi) = \frac{y_i\underline{z}_i^t\underline{\beta} - b(\underline{z}_i^t\underline{\beta})}{\phi} + c(y_i,\phi)$$

Como las observaciones son independientes, el logaritmo de la verosimilitud corresponde a la suma de las contribuciones de cada observación:

$$l(\underline{\beta}) = \sum_i l_i(\underline{\beta})$$

Teniendo ya la función de verosimilitud, los estimadores de máxima verosimilitud de los parámetros desconocidos son las soluciones de igualar la primera derivada de la función de verosimilitud a cero:

$$s(\underline{\beta}) = \frac{\partial l}{\partial \underline{\beta}} = 0$$

Usualmente, las ecuaciones que tenemos no son lineales por lo que deberán de resolverse mediante métodos numéricos. Uno de los métodos que se suele utilizar en estos casos es el método iterativo de mínimos cuadrados ponderados.

2.3. Bondad de ajuste del modelo

En la práctica, suele ocurrir que existen diversos modelos teóricos, que se pueden ajustar a unos mismos datos, queremos que el elegido se ajuste lo mejor posible a ellos, y esto lo podremos comprobar viendo las diferencias existentes entre los valores observados y los valores esperados en el modelo de estudio. Existen pruebas que miden estas discrepancias, entre ellas están los siguientes estadísticos:

- Desviación (Deviance):

$$D = \sum_{i=1}^{n} 2[y\theta(y_i) - \theta(\mu_i) - b\theta(y_i) + b\theta(\mu_i)]$$

- χ^2 de Pearson:

$$\chi^2 = \sum_{i=1}^{n} \frac{(y_i - \mu_i)^2}{Var(\mu_i)}$$

Donde $Var(\mu_i)$ es la varianza estimada para el modelo que se esté considerando.

El modelo que mejor se ajusta a los datos será en el que obtengamos menor valor del estadístico.

Capítulo 3

Regresión de Poisson

Una vez explicados los Modelos Lineales Generalizados, estudiaremos dos casos particulares de estos, el modelo de regresión de Poisson y el modelo de regresión Binomial Negativa, pues son los más apropiados para trabajar con datos de conteo como los que obtenemos a través de RNA-Seq. Primero se estudiará el modelo de regresión de Poisson y sus características más importantes, en especial la propiedad de *equidispersión* que deben de presentar los datos que se ajusten a través de esta distribución, lo que nos llevará a proponer el ajuste de un modelo de regresión Binomial Negativa. En este capítulo nos centraremos en el estudio del Modelo de Regresión de Poisson. Podemos obtener más información de esta regresión en Hardin and Hilbe [4].

3.1. Modelo de Regresión de Poisson

Realmente el modelo de regresión de Poisson, es un MLG en el cual la variable respuesta Y sigue una distribución de Poisson de media μ, con $\mu > 0$, lo que se denota como $Y \sim Po\ (\mu)$.

Esta distribución se suele utilizar para modelar la probabilidad de que durante un determinado periodo de tiempo ocurra un cierto número de eventos, por ejemplo, el número de personas que llaman a una centralita de teléfono en un periodo de tiempo determinado. También se suele utilizar para modelar datos de conteo en un cierto espacio, por ejemplo, el número de errores que comete una fotocopiadora en dos impresiones. Se podría decir que la distribución de Poisson corresponde a datos de conteos en la misma forma que la distribución Normal lo es para datos continuos. La función de probabilidad de esta distibución es:

$$P(Y = y|\mu) = \frac{e^{-\mu}\mu^y}{y!}; \quad \text{para} \quad y \in \mathbb{Z}_0^+;\ \mu > 0$$

Y su media y varianza son:

$$E(Y) \ = \ Var(Y) \ = \ \mu$$

Como se muestra, la distribución de Poisson, es una distribución discreta y que cumple la propiedad de equidispersión, es decir, que la varianza y la media son iguales. Por esta propiedad, se tiene que mientras mayor sea el valor esperado, mayor dispersión tendrán los valores que puede tomar la variable que se distribuya así.

3.1.1. Modelo de Regresión de Poisson como un MLG

Si tomamos logaritmo neperiano de esta distribución se puede comprobar que este modelo pertenece a la familia exponencial, y además podremos identificar los elementos necesarios que nombramos en la Seccion 2.1:

$$ln\ P[Y = y] = -\mu + yln(\mu) - ln(y!) = yln(\mu) - \mu - ln(y!)$$

Los elementos serían:

- La función link, es:
$$\theta = \theta(\mu) = ln(\mu)$$
- El parámetro de dispersión:
$$\phi = 1$$
- Función cumulante:
$$b(\theta) = \mu$$
- Función c(y,ϕ), específica de cada distribución, como se había explicado en MLG:
$$c(y, \phi) = ln(y!)$$

Como se puede ver, es un modelo que pertenece al MLG, ahora, una vez que hemos identificado estos elementos, vamos a estudiar el modelo de regresión.

Para obtener la función que predice la media en este modelo vamos a seguir los siguientes pasos:

- Obtenemos μ a partir de la función link:
$$\mu = e^{\theta}$$
- Puesto que $\theta(\mu) = ln(\mu)$:
$$ln(\mu) = \beta_0 + \beta_1 x_1 + ... + \beta_k x_k$$
- La función que predice la media sería:
$$\mu = e^{\nu}; \qquad donde\ \nu = \beta_0 + \beta_1 x_1 + ... + \beta_k x_k$$

3.1.2. Interpretación de los coeficientes

Veamos la interpretación de los coeficientes considerando sólo una variable independiente X, es decir se predice:

$$\theta^* = \beta_0 + \beta_1 X$$

Suponiendo que se han observado los valores de X, x_1 y x_2 para los individuos 1 y 2, respectivamente, se tendría:

$$\begin{aligned} \theta_1^* &= \theta^*(X = x_1) = \beta_0 + \beta_1 x_1 \\ \theta_2^* &= \theta^*(X = x_2) = \beta_0 + \beta_1 x_2 \end{aligned}$$

La diferencia entre las predicciones sería:

$$\theta_2^* - \theta_1^* = \beta_1(x_2 - x_1)$$

Teniendo en cuenta la expresión de la función link en este modelo, $\theta = ln(\mu)$, se obtiene que:

$$\begin{aligned} ln(\mu_2^*) - ln(\mu_1^*) = \beta_1(x_2 - x_1) &\Leftrightarrow ln\left(\frac{\mu_2^*}{\mu_1^*}\right) = \beta_1(x_2 - x_1) \\ &\Leftrightarrow \frac{\mu_2^*}{\mu_1^*} = e^{\beta_1(x_2 - x_1)} \end{aligned}$$

En el caso particular en que el predictor aumente en una unidad, se tiene que:

$$\frac{\mu^*(x+1)}{\mu^*(x)} = e^{\beta_1} \Leftrightarrow \mu^*(x+1) = \mu^*(x)e^{\beta_1}$$

Se puede observar de esta manera que el aumento de una unidad en la variable X es múltiplo de la media anterior, siendo el factor de proporcionalidad e^{β_1}.

3.2. El problema de la sobredispersión

Normalmente, los paquetes estadísticos ofrecen estimadores de $\underline{\beta}$ que maximizan la función de verosimilitud. Pero para obtener inferencias válidas respecto de este parámetro, necesitamos verificar el supuesto de que la media y varianza condicionales para este modelo son iguales, es decir, el supuesto de equidispersión. Se ha demostrado que, aunque este supuesto no se cumpla (lo cual sucede muchas veces), el estimador puntual de $\underline{\beta}$ si será válido, pero no lo será el estimador de su error estándar, y por tanto las inferencias respecto de $\underline{\beta}$ tampoco lo serán.

Esta situación se debe a que en algunas ocasiones hay correlación entre las observaciones, falta de homogeneidad en el material experimental, existencia de datos atípicos o exceso de ceros, generando un fenómeno llamado sobredispersión, o dicho de otra manera, cuando la variabilidad de los datos es mayor que el valor estimado

para ellos, es decir, cuando Var[X] > E[X]. Esto implica la sobrestimación de los errores estándar, lo que a su vez provoca una subestimación de la significación estadística de los coeficientes de los parámetros, llevando a que algunos coeficientes sean considerados significativos cuando realmente no lo son.

Por ello, aunque la regresión de Poisson sea la más común para datos de conteos, se han propuesto algunas alternativas que suavizan el supuesto de que la media y la varianza son iguales. Una de las alternativas es la regresión Binomial Negativa, la cual utilizaremos en el estudio para datos procedentes de RNA-Seq y que se explicará en el capítulo siguiente.

Existen diversos métodos para detectar la sobredispersión aunque generalmente se detecta evaluando la relación entre los estadísticos χ^2 de Pearson o la función *deviance*, D, definidos en la Sección 2.3 y sus respectivos grados de libertad (gl), es decir evaluando:

$$\frac{\chi^2}{gl} \quad y \quad \frac{D}{gl}$$

En el caso en que estos valores sean mayor que 1, nos indican la existencia de sobredispersión

Capítulo 4

Regresión Binomial Negativa

Se propone como una forma de resolver el problema de la "sobredispersión" que surge frecuentemente al aplicar un modelo de regresión de Poisson a datos de RNA-Seq. En la literatura existen diferentes formas de introducir la Binomial Negativa como puede verse en los textos de Hilbe [5].
El software con el que trabajamos puede considerarse variaciones del denominado "Modelo de regresión Binomial Negativa con sobredispersión variable", cuya base teórica exponemos a continuación.

4.1. Modelo de Regresión Binomial Negativa

Se puede obtener siguiendo dos métodos diferentes:

1. Derivación en términos de una mixtura Poisson-Gamma.
2. Derivación en términos de la expresión clásica de la función de probabilidad de la Binomial Negativa.

En este contexto tiene sentido ver el primero de los métodos anteriormente señalados.

4.1.1. Binomial Negativa como mixtura Poisson-Gamma

Suponemos que la variable respuesta Y_i sigue una ditribución de Poisson con media μ_i, es decir:

$$Y_i \sim Po\,(\mu_i); \qquad donde \;\; \mu_i = \lambda_i u_i$$

$ln\mu_i$ se modeliza como un modelo lineal:

$$ln\;\mu_i \;=\; ln\lambda_i \;+\; lnu_i \;=\; \underline{X_i'}\underline{\beta} \;+\; \epsilon_i$$

Donde u_i es el efecto observado, error o ruido para el que se propone una distribución Gamma de media 1:

$$u_i \ \sim \ Ga\ (\nu, \nu); \quad por\ tanto\ E[u_i] = \frac{\nu}{\nu} = 1; \ \ Var[u_i] = \frac{\nu}{\nu^2} = \frac{1}{\nu}$$

Con esta notación se tiene que:

$$Y_i|_{\lambda_i, u_i} = Y_i|_{\underline{x}_i, u_i} \ \sim \ Po\ (\mu_i) \quad donde \ \ \mu_i = \lambda_i u_i, \ \ u_i \ \sim \ Ga\ (\nu, \nu)$$

y la función de probabilidad de Y_i se puede probar que es:

$$f_{Y_i}(y_i) = \frac{\Gamma\left(y_i + \frac{1}{\alpha}\right)}{\Gamma(y_i + 1)\ \Gamma\left(\frac{1}{\alpha}\right)} \left(\frac{1}{1 + \alpha\mu_i}\right)^{1/\alpha} \left(1 - \frac{1}{1 + \alpha\mu_i}\right)^{y_i} \tag{4.1}$$

con $y_i \in \mathbb{Z}_0^+, \ \alpha = 1/\nu$.

Puede verse en Hilbe [5] que (4.1) corresponde a la función de probabilidad de una distribución Binomial Negativa con características:

- $E[Y] = \mu$
- $Var[Y] = \mu + \alpha\mu^2$
- Se define el índice de sobredispersión, cociente varianza-media, o coeficiente biológico de variación (BCV) como:

$$BCV = \frac{Var[Y]}{E[Y]} = 1 + \alpha\mu \tag{4.2}$$

Observación:
Puede verse en (4.2) que el modelo de regresión Binomial Negativo puede considerarse una generalización de la regresión de Poisson, ya que si $\alpha = 0$ tendríamos el modelo de regresión de Poisson.

4.1.2. Características del Modelo de Regresión Binomial Negativa como un MLG

De la expresión de la función de probabilidad dada en (4.1) se deduce que:

- La función vínculo es:

$$\theta \ = \ ln\ \left(\frac{\alpha\mu}{1 + \alpha\mu}\right)$$

- El parámetro de dispersión de la familia exponencial en forma canónica:

$$\phi \ = \ 1$$

- La función cumulante:

$$b(\theta) \;=\; -\frac{1}{\alpha}\; ln\; \left(\frac{1}{1+\alpha\mu}\right)$$

4.1.3. Interpretación de los coeficientes

Considerando el estudio de una sola variable independiente X, y suponiendo que se han observado los valores x_1 y x_2 de dicha variable tenemos que la razón que mide el cambio de un valor observado a otro es el mismo que para el modelo de regresión de Poisson estudiado en la Sección 3.1.2.
En el caso particular en que el predictor aumente en una unidad, al igual que en el modelo de Poisson, se tendría que:

$$\mu^*(x+1) = \mu^*(x)e^{\beta_1}$$

Puede verse más información sobre este punto en Hilbe [5].

4.1.4. Modelo de Regresión Binomial Negativa cuando se reduce a la comparación de dos grupos independientes

Cuando comparamos dos grupos independientes, es como si se considerara el modelo anterior con sólo una variable independiente, X, siendo ésta una variable indicadora, definida de la siguiente forma:

$$X = \begin{cases} 1, & \textit{si un individuo pertenece al grupo tratado} \\ 0, & \textit{si un individuo pertenece al grupo control} \end{cases}$$

El estudiar la diferencia de estar en un grupo u otro, es equivalente a estudiar el cambio que sufre la variable respuesta al aumentar la variable independiente en una unidad, explicado en la Sección 3.1.2. Pues si $x_B = 0$, y estudiamos el cambio al aumentar una unidad esta variable, entonces $x_A = x_B + 1 = 1$, lo que en la comparación entre grupos significará el cambio de un grupo a otro.
Si recordamos, en la Sección 3.1.2, obtuvimos como resultado que el logaritmo neperiano del cociente de medias puede expresarse como:

$$ln\left(\frac{\mu_A^*}{\mu_B^*}\right) = b_1(x_A - x_B) = b_1(1-0) = b_1$$

Por tanto, tenemos que b_1 proporciona la información sobre la consecuencia de pertenecer a un grupo u otro.

A la razón entre las medias, $\frac{\mu_A^*}{\mu_B^*}$ se le denomina generalmente *fold-change*. En la literatura es usual trabajar con el logaritmo en base 2 de este cociente. En este caso la relación entre las medias debe interpretarse como sigue:

$$Logaritmo\ en\ base\ dos:\ log_2\left(\frac{\mu_A^*}{\mu_B^*}\right) = b_1 \Rightarrow \mu_A^* = \mu_B^*\ 2^{b_1}$$

Obsérvese que esta relación es análoga a la que tendríamos utilizando un modelo de Regresión Binomial Negativa:

$$Logaritmo\ neperiano:\ ln\left(\frac{\mu_A^*}{\mu_B^*}\right) = b_1 \Rightarrow \mu_A^* = \mu_B^*\ e^{b_1}$$

Aún así, hoy en día, se sigue utilizando $log_2 fold-change$ para estudiar el efecto de pertenecer a un grupo u otro, pero las consideraciones anteriores deben ser tenidas en cuenta.

4.2. Comparación de la expresión diferencial de genes en dos grupos independientes.

Cuando se estudia la expresión diferencial de genes en dos grupos independientes (denominados A y B), el contraste que realizaremos, realmente se reduce a una comparación de medias, como se explicó en la Sección 1.1, por lo que las hipótesis planteadas para un gen g tendrá la forma:

$$H_0^g : \mu_A^g = \mu_B^g$$
$$H_1^g : \mu_A^g \neq \mu_B^g$$

Donde $\mu_A^g\ y\ \mu_B^g$ corresponden a las medias de los conteos del gen g en las condiciones A y B, respectivamente.

El contraste anterior se suele formular en términos de *fold-change* definido en la sección anterior como:

$$fold - change = \frac{\mu_A}{\mu_B}$$

Pudiendo tomar este parámetro los siguientes valores:

$$fold - change = \begin{cases} > 1 & si\ \mu_A > \mu_B \\ = 1 & si\ \mu_A = \mu_B \\ < 1 & si\ \mu_A < \mu_B \end{cases}$$

Nota: Obsérvese que:

- Si $\mu_B = 0$ entonces el *fold-change* es infinito.

- Si $\mu_A = \mu_B = 0$ entonces tendremos una indeterminación.

Estos casos se tratarán en más detalle en los ejemplos prácticos.

Más aún, debido a la magnitud y gran variabilidad de los datos de conteo que se tienen en los estudios de RNA-Seq, no se trabaja directamente con el *fold-change*, sino con el logaritmo en base dos de este parámetro, que ya habíamos explicado anteriormente su procedencia.

$$log_2 fold - change = log_2 \left(\frac{\mu_A}{\mu_B} \right) = log_2 \mu_A - log_2 \mu_B$$

Observación: En la siguiente figura se ha representado la función logaritmo en base 2.

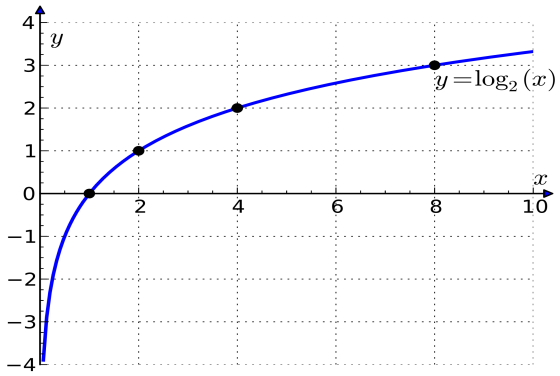


Figura 4.1: Función logaritmo en base 2.

Esta figura muestra que, mientras mayores valores de x se tengan, menor crecimiento de la y se va a observar. Esto se debe a que la función logarítmica tiende de alguna forma a casi estabilizarse, cada vez va a ir creciendo menos hasta que prácticamente su crecimiento sea nulo. Por eso al aplicar logaritmo al *fold-change* tendremos que por muy elevados que sean los conteos de algunos genes, el $log_2 fold - change$ no presentará valores tan elevados como estos.

Se tienen las siguientes relaciones para interpretar los valores de este parámetro:

$$log2fold - change = \begin{cases} > 0, & si\ \mu_A > \mu_B \\ = 0, & si\ \mu_A = \mu_B \\ < 0, & si\ \mu_A < \mu_B \end{cases}$$

Por tanto, el contraste planteado inicialmente para la comparación de los conteos medios del gen g en los grupos A y B sería:

$$H_0^g : \mu_A^g = \mu_B^g$$
$$H_1^g : \mu_A^g \neq \mu_B^g$$

es equivalente al contraste

$$H_0^g : log_2\left(\frac{\mu_A^g}{\mu_B^g}\right) = 0$$
$$H_1^g : log_2\left(\frac{\mu_A^g}{\mu_B^g}\right) \neq 0$$

y este último a su vez a

$$H_0^g : log_2(\mu_A^g) - log_2(\mu_B^g) = 0$$
$$H_1^g : log_2(\mu_A^g) - log_2(\mu_B^g) \neq 0$$

Existen diversos estadísticos que se pueden utilizar para proceder con este constraste. En nuestro estudio nos basaremos en el *estadístico de Wald*:

$$t_g = \frac{log_2(\hat{\mu}_A) - log_2(\hat{\mu}_B)}{\sqrt{\widehat{Var}[log_2(\hat{\mu}_A) - log_2(\hat{\mu}_B)]}} \tag{4.3}$$

cuya distribución asintótica es *N(0,1)*.

Observación: Recordemos que para un contraste de la forma:

$$H_0 : \theta = \theta_0$$
$$H_1 : \theta \neq \theta_0$$

El estadístico de Wald se define como:

$$T = \frac{(\hat{\theta} - \theta_0)^2}{\widehat{Var}(\hat{\theta})} \simeq \chi_1^2$$

ó equivalentemente como

$$T = \frac{\hat{\theta} - \theta_0}{\sqrt{\widehat{Var}(\hat{\theta})}} \simeq N(0,1)$$

En (4.3), la $Var[log_2(\hat{\mu}_A) - log_2(\hat{\mu}_B)]$ se estima utilizando el método delta (siempre que μ_A y μ_B sean distintas de cero).

$$\begin{aligned}\widehat{Var}[log_2(\hat{\mu}_A) - log_2(\hat{\mu}_B)] &= \widehat{Var}[log_2(\hat{\mu}_A)] + \widehat{Var}[log_2(\hat{\mu}_B)] \\ &= \left(\frac{1}{\hat{\mu}_A^g}\right)^2 \widehat{Var}(\hat{\mu}_A^g) + \left(\frac{1}{\hat{\mu}_B^g}\right)^2 \widehat{Var}(\hat{\mu}_B^g)\end{aligned}$$

Puesto que tenemos un estadístico con distribución *N(0,1)* la región crítica y el p-valor para los contrastes se calculan en la forma usual para estos tipos de test.

Capítulo 5

Comparaciones múltiples

Cuando el número de comparaciones posibles es elevado, la aplicación reiterada de un test de comparación simple, para un nivel de significación α dado, puede conducir a un número grande de rechazos de la hipótesis nula aunque no existan diferencias reales. El intento de solucionar el problema de los falsos rechazos justifica el estudio de los métodos para comparaciones múltiples.
En este capítulo estudiaremos para esta situación, los tipos de errores que se suelen tratar de controlar a la hora de realizar un contraste de hipótesis y métodos apropiados para cada tipo de error. En nuestro caso, el análisis de la expresión diferencial de genes, tenemos muchos contrastes que realizar, uno por cada gen, y como ya se ha mencionado anteriormente, en un estudio de este tipo suelen existir miles de genes, por lo que los métodos de comparaciones múltiples suelen ser necesarios. Puede verse más información sobre esto en A.S. Foulkes [6].

5.1. Tipo de errores a controlar

Recordemos que son dos los posibles errores que se pueden cometer al realizar un test de hipótesis, el primero sucede cuando el test resulta significativo, por lo que se rechaza la hipótesis nula y en realidad es cierta, al que se le denomina tradicionalmente como error de tipo uno (ETI) y el segundo sucede cuando el test nos lleva a no rechazar la hipótesis nula y esta es falsa, denominado como error de tipo dos ($ETII$).

	Test	
Realidad	No significativo	Significativo
H_0 cierta	Decisión correcta	Error de tipo I
H_1 cierta	Error tipo II	Desición correcta

Cuadro 5.1: Errores en contrastes de hipótesis.

Tradicionalmente, las pruebas estadísticas se basan en el control de la tasas de ETI, para ello al realizar un test se fija un nivel se signicación, o nivel del test, α,

normalmente con un valor de 0.05, que mide la probabilidad de rechazar incorrectamente la hipótesis nula, es decir, el nivel de un test es la probabilidad de obtener un ETI. Usualmente el p-valor, definido como la probabilidad de observar un resultado igual o más extremo que el estadístico del test, se compara con α para tomar la decisión de rechazar o no la hipótesis nula.

Si se considerara ahora el problema de contrastar m hipótesis nulas dadas por $H_0^1, ..., H_0^m$, en cada una de las m pruebas podrían ocurrir los dos tipos de errores recogidos en el Cuadro 5.1. Sumando sobre todas las pruebas se obtienen los siguientes resultados:

	Test		
Realidad	No significativo	Significativo	
H_0 cierta	U	V	m_0
H_1 cierta	T	S	$m - m_0$
	m - R	R	m

Cuadro 5.2: Errores en contrastes de hipótesis múltiples.

U y S corresponden al número de veces que se han tomado decisiones correctas en los m contrastes, V es el número de ETI cometidos y T el número de ETII.

La gran parte de los métodos de ajuste para comparaciones múltiple recurren a las tasas **FWER** (family-wise error rate) y **FDR** (false discovery rate), las cuales, al igual que en los tests de hipótesis simples, tratan de controlar la tasa de ETI o lo que es lo mismo, tratan de controlar V.

5.2. FWER (Family-wise error rate)

Esta tasa de error se define como la probabilidad de cometer al menos un ETI, por lo que tenemos que:

$$FWER = Pr\ (V \geq 1)$$

La FWER se puede definir con mayor precisión en términos de si la hipótesis nula es cierta.

5.2.1. Método Bonferroni para controlar el FWER.

El método más conocido para controlar el FWER es el de Bonferroni. Por este método, si se quiere tener un nivel global máximo de α, se debe tomar para el contraste de cada gen un nivel corregido $\alpha^* = \alpha/m$, siendo m el número total de contrastes o equivalentemente el número de genes existentes.

Ejemplo 1.
Queremos detectar diferencias en 10 genes. Para cada gen tendremos una hipótesis nula a contrastar. Supongamos que los p-valores obtenidos ordenados de menor a mayor son los siguientes:

$$p_{(1)} = 0.001\ ;\ p_{(2)} = 0.012\ ;\ p_{(3)} = 0.014\ ;\ p_{(4)} = 0.122\ ;\ p_{(5)} = 0.245\ ;$$
$$p_{(6)} = 0.320\ ;\ p_{(7)} = 0.550\ ;\ p_{(8)} = 0.776\ ;\ p_{(9)} = 0.840\ ;\ p_{(10)} = 0.995$$

El ajuste de Bonferroni nos llevaría a usar el nivel de ajuste significativo de $\alpha^* = 0{,}05/10 = 0{,}005$. *Basándonos en esto, solo rechazaríamos* $H_0^{(1)}$, *pues solo el primer p-valor cumple la condición de ser menor que* α^*.

Esta propuesta, da una cota para el nivel global. Puede ser demasiado conservativa y cuando la cantidad de contrastes es grande los niveles corregidos resultan demasiado bajos. Esto significa que se seleccionarán pocos genes como candidatos a estar DE.

5.3. FDR (False discovery rate)

Esta tasa o razón ha ganado popularidad en la última decada como medida de error asociada a contrastes de hipótesis. Formalmente, FDR se define como la proporción esperada de hipótesis nulas que son verdaderas entre las que son declaradas como significativas.

Volviendo a la notación definida en el Cuadro 5.2, asumiendo que R, número de pruebas que han resultado significativas, es mayor que 0, FDR se define como:

$$FDR = E\left(\frac{V}{R}\right) \qquad \textit{donde}\ \ E(\cdot)\ \textit{es la esperanza matemática}$$

Para $R = 0$, se define la razón $\frac{V}{R}$ como 0 ya que si no existen pruebas significativas no pueden ocurrir falsos rechazos. Por tanto FDR puede expresarse como:

$$\begin{aligned} FDR &= E\left(\frac{V/R}{R>0}\right) Pr(R>0) + E\left(\frac{V/R}{R=0}\right) Pr(R=0) \\ &= E\left(\frac{V/R}{R>0}\right) Pr(R>0) \end{aligned}$$

Bajo el supuesto de que todas las hipótesis nulas son ciertas, En el Cuadro 5.2, tendríamos que $V = R$ y V/R será:

$$\frac{V}{R} = \begin{cases} 0 & si\ V = 0 \\ 1 & si\ V \geq 1 \end{cases}$$

Entonces:

$$\begin{aligned} FDR &= E\left(\frac{V}{R}\right) = 0 * Pr(V = 0) + 1 * Pr(V \geq 1) \\ &= Pr(V \geq 1) = FWER \end{aligned}$$

Esto significa, que si todas las hipótesis nulas son ciertas, entonces la FDR es igual a FWER.

Si se supusiera ahora que no todas las hipótesis nulas son ciertas, entonces se tendría que V < R, por lo que $\frac{V}{R} < 1$ y:

$$\begin{aligned} FDR &= E\left(\frac{V}{R}\right) = \left(\frac{V}{R}\right) * Pr(V \geq 1) + \left(\frac{0}{R}\right) * Pr(V = 0) \\ &= \left(\frac{V}{R}\right) * Pr(V \geq 1) < Pr(V \geq 1) = FWER \end{aligned}$$

$$\text{Es decir:} \quad FDR < FWER$$

Como consecuencia, se tiene el resultado general de que FWER es menor o igual que FDR, lo que implica que cualquier enfoque que controle FWER también controlará FDR. El contrario, sin embargo, no es cierto.

La elección de la medida de error a usar se basa en gran medida en el objetivo científico y las expectativas de la investigación. Cuando existen un gran número de variables para analizar, cada vez es más frecuente controlar el FDR, esto se debe, en gran parte, a que se espera que muchas de las correspondientes hipótesis sean falsas. Si el número de hipótesis nulas verdaderamente falsas es pequeño o la consecuencia de equivocarse por la prueba significativa es grave, entonces la FWER puede ser una medida más apropiada.

5.3.1. Método de Benjamini y Hochberg (B-H) para controlar el FDR.

Como se describió en la Sección 5.3, el FDR es una medida atractiva de la tasa de error en el contexto de las investigaciones genéticas. Uno de los enfoques más utilizados para controlar el FDR, es el ajuste de Benjamini y Hochberg (B-H), propuesto por Benjamini y Hochberg (1995), basado en el supuesto de independencia de los p-valores. A diferencia del método de Bonferroni este método considera un nivel de significación diferente para cada contraste.

Vamos a considerar contrastar la serie de hipótesis nulas independientes dadas por $H_0^1, ..., H_0^m$, donde se han obtenido los p-valores $p_1, ..., p_m$ y además, supongamos que queremos controlar el FDR a un nivel q. Los pasos que sigue el método B-H son los siguientes:

- Sea $p_{(1)}, ..., p_{(m)}$ los p-valores observados ordenados de tal manera que

$$p_{(1)} \leq ... \leq p_{(m)}$$

y sean las correspondientes hipótesis nulas dadas por $H_0^{(1)}, ..., H_0^{(m)}$

- Se define k como

$$k = max \left\{ i : p_{(i)} \leq \frac{i}{m} q \right\} \tag{5.1}$$

- Rechazamos las hipótesis nulas $H_0^{(1)}, H_0^{(2)}, ..., H_0^{(k)}$

El procedimiento es más fácil de lo que parece, se verá más claro con un ejemplo.

Ejemplo 2.
Vamos a continuar con el Ejemplo 1 donde por el método de Bonferroni solo rechazamos $H_0^{(1)}$. Recordemos que los p-valores observados ordenados eran:

$$p_{(1)} = 0.001 \ ; \ p_{(2)} = 0.012 \ ; \ p_{(3)} = 0.014 \ ; \ p_{(4)} = 0.122 \ ; \ p_{(5)} = 0.245 \ ;$$
$$p_{(6)} = 0.320 \ ; \ p_{(7)} = 0.550 \ ; \ p_{(8)} = 0.776 \ ; \ p_{(9)} = 0.840 \ ; \ p_{(10)} = 0.995$$

Usando el método de B-H, los niveles de significación para cada contraste, $\alpha_i^* = 0{,}05 \left(\frac{i}{10}\right)$, vienen dados por:

$$\alpha_1^* = 0.005 \ ; \ \alpha_2^* = 0.010 \ ; \ \alpha_3^* = 0.015 \ ; \ \alpha_4^* = 0.020 \ ; \ \alpha_5^* = 0.025 \ ;$$
$$\alpha_6^* = 0.030 \ ; \ \alpha_7^* = 0.035 \ ; \ \alpha_8^* = 0.040 \ ; \ \alpha_9^* = 0.045 \ ; \ \alpha_{10}^* = 0.050$$

El valor de la función (5.1) es simplemente el máximo i para el cual $p_{(i)}$ es menor o igual a α_i^*. En este ejemplo, tenemos $k = 3$ pues $p_{(3)} = 0{,}014 < 0{,}015$, mientras $p_{(i)} > \alpha_i^*$ para $i > 3$.

Entonces por B-H rechazaríamos $H_0^{(1)}$, $H_0^{(2)}$ y $H_0^{(3)}$. El hecho de que $p_{(2)} = 0{,}012$ no sea menor que $\alpha_i^* = 0{,}010$ no es relevante, ya que el siguiente p-valor $p_{(3)} = 0{,}014$ sí cumple con el criterio de rechazo.

Además de definir los criterios de rechazo de cada contraste, también podemos calcular los p valores ajustados. El procedimiento para este cálculo es el siguiente:

1. Primero calculamos un p valor ajustado para cada i de la siguiente forma:

$$p_{(i)}^{adj} = p_{(i)} \frac{m}{i}$$

2. Entonces actualizamos estos valores de p para asegurar monotonicidad dejando:

$$p_{(i)}^{adj} = min_{(j \geq i)} \left(p_{(i)}^{adj} \right)$$

Durante los casos prácticos que se desarrollarán en las Secciones 6.1.1 y 6.2.1, se utilizará este método, pues al trabajar con una cantidad enorme de genes el método de Bonferroni es demasiado restrictivo y nos propondrá menos genes DE que este otro método por el simple hecho de tener un enfoque más conservador. Puede verse más información sobre el ajuste propuesto por Benjamini and Hochberg en [7].

Capítulo 6

Paquetes en R y Bioconductor

R, desarrollado inicialmente por Robert Gentleman y Ross Ihaka (1993), es un lenguaje y entorno de programación para análisis estadístico y gráfico. Probablemente, uno de los más utilizados por la comunidad estadística. Posee una gran variedad de bibliotecas o paquetes sobre diversas finalidades. Por otro lado, **Bioconductor** proporciona las herramientas necesarias para el análisis y la comprensión de los datos genómicos de alto rendimiento. Utiliza el lenguaje de programación estadística R, y es de código y desarrollo abierto.

R y Bioconductor disponen de numerosos paquetes a través de los cuales podemos realizar un análisis de datos RNA-Seq. La elección del paquete adecuado dependerá, en gran medida, del número de réplicas biológicas disponibles en el ensayo, es decir, del número de individuos o muestras que tengamos en cada condición estudiada.

En especial, se van a estudiar en profundidad tres paquetes, el paquete **DESeq2**, para el cuál necesitaremos del estudio, podríamos decir paralelo, del paquete anterior a este, **DESeq**, y el paquete **edgeR**. Para los tres paquetes se necesitan los mismos datos de entrada, una matriz con los conteos brutos y factores con información sobre las muestras. El proceso es similar en los tres paquetes, sin embargo cada uno tiene sus peculiaridades. Tras la explicación de cada paquete se realizará un ejemplo práctico utilizándolos.

Para poder trabajar con Bioconductor tenemos que cargarlo cuando abramos R:

```
# Buscar Bioconductor en la web:
> source("http://bioconductor.org/biocLite.R")

# Instalar la librería Biobase:
> biocLite("Biobase")

# Cargar la librería Biobase:
> library(Biobase)
```

6.1. Paquete edgeR

edgeR es un paquete escrito para R que realiza expresión diferencial genética utilizando datos de conteos bajo un modelo binomial negativo. Se puede obtener más información acerca de este paquete en la página web de Bioconductor:

http://www.bioconductor.org/packages/2.8/bioc/html/edgeR.html

A continuación se explicarán las funciones más importantes del paquete y el procedimiento para realizar el análisis de expresión diferencial con el mismo, luego se realizará un ejemplo práctico para afianzar los conocimiento y explicar las salidas que nos proporcionan las funciones.

Como guía del estudio de este paquete se utilizará el manual que se puede encontrar en la página de Bioconductor, sin embargo actualmente existe una revisión de este manual más reciente, exactamente del 5 de Octubre de 2015, que es la que nos ofrece la página de Bioconductor actualmente. Sin embargo cuando se comenzó a trabajar con este paquete se utilizó la versión anterior, ya que aún esta no había salido, por lo que se pueden ver pequeñas alteraciones.

Clase DGEList

Cada paquete tiene sus propias clases o condiciones para poder trabajar con él, este paquete necesita la clase *DGEList* para poder realizar cualquier análisis. Para crearla se utiliza la propia función *DGEList*:

DGEList(counts = matrix(0, 0, 0), lib.size = colSums(counts), norm.factors = rep(1,ncol(counts)), group = rep(1,ncol(counts)), remove.zeros = FALSE)

En el atributo *count* irá la matriz con los conteos brutos, donde se debe de tener por filas los genes y por columnas las muestras o individuos, como se explicó en la Sección 1.1, en *group* la condición de cada muestra bajo estudio, por ejemplo la condición tratados o no tratados que corresponda a cada muestra, y por último en *gene* se puede incluir algún vector o data.frame que indique alguna información de los genes, no es necesario incluir este atributo, de hecho, por defecto será *genes = NULL*. A través del atributo *remove.zeros* se pueden eliminar los genes que tienen siempre conteos nulos, es decir, que en todas las columnas, la fila que corresponde a ese gen tiene valores ceros. Este atributo no lo cambiaremos, pues como ya se explicará en el siguiente apartado, se realizará una depuración de los datos que no solo eliminará estos genes sino también los que se expresan con muy baja frecuencia.

Una vez que se tenga el objeto con la clase *DGEList* se puede comenzar con el análisis. Se podrán hacer estudios estadísticos de este nuevo objeto al igual que se podían hacer antes a la matriz conteo, pues no se ha perdido ninguna información.

Filtrado de los genes

Es muy importante, antes de comenzar con el análisis diferencial de los genes, eliminar aquellos que no tienen conteos, o que apenas tienen. Pues de antemano se sabe que estos genes no se van a expresar de forma distinta en las diferentes condiciones que se estén estudiando, ya que al no tener apenas conteos para ninguna condición su estudio será nulo. Por ejemplo, si tuvieramos un gen que no se ha expresado, es decir, que ni en una condición ni en otra ha tenido conteos, al realizarle el estudio para saber si es diferencialmente expresado en las debidas condiciones, como resultado tendremos que no lo es, pues en todas las condiciones ha tenido los mismos conteos, es decir, cero. Pero eso se sabe de antes de realizarle el estudio, pues si no ha tenido conteos no existen datos que analizar.

Realmente se trata de una depuración de datos como se haría en cualquier estudio estadístico, donde se analiza que hacer con aquellos datos que se identifican como outlier o que representan algún tipo de problema al estudio.

Entonces, ya que su estudio no nos servirá, lo correcto sería eliminarlos, pues aunque no nos proporcione resultados útiles, a la hora de representarlos, por ejemplo, tendríamos muchos menos y podríamos tener un gráfico más claro. Al igual que en los gráficos, ocurrirá en los estudios estadísticos o en la tabla final donde aparezca el resultado del análisis diferencial, ya que al tener menos genes también se tendrían menos filas. Para eliminar estos genes basta con identificar aquellos que no tienen apenas conteos y eliminarlos del objeto *DGEList* creado y ya se podrá seguir con el estudio.

Normalización

Este paquete, por defecto, trabaja la normalización de los conteos brutos a través del método *TMM* explicado en la Sección 1.4. En **edgeR** existe una función que calcula unos factores de normalización para escalar el tamaño bruto de las librerías o muestras:

calcNormFactors(object, method=c("TMM","RLE","upperquartile","none"), ...)

En este caso solamente se indicará el atributo *object*, donde irá el objeto de clase *DGEList* para normalizarlo, los demás atributos se pueden dejar con los valores asignados por defecto.

Estimación de la dispersión

Este paso es de suma importancia en cualquiera de los paquetes con los que trabajemos y cada uno de ellos tiene una forma propia de estimar las dipersiones de cada gen.

Al igual que en los paquetes **DESeq** y **DESeq2**, explicados posteriormente, el procedimiento de cálculo de estas dispersiones consta de 3 pasos, sin embargo cada

paquete tiene sus propias instrucciones y métodos, por lo que en ninguno de los tres se obtendrán las mismas dispersiones para ningún gen.

Este paquete calcula la dispersión común o variabilidad total para todos los genes mediante el método **qCML**. Para ello emplea la función **estimateCommonDisp(·)**:

estimateCommonDisp(object, verbose=FALSE,...)

En *object* se dará el objeto creado anteriormente con la clase *DGEList*, y el atributo *verbose* sirve para que muestre, tras aplicar la función, la dispersión común estimada y el coeficiente de variación biológica, que es la raíz cuadrada de la dispersión.

Sin embargo para genes con bajo nivel de expresión la dispersion común estimada supone una mayor variabilidad de la que realmente presentan, por ello es recomendable estimar la dispersión gen a gen o dispersión tagwise, donde a partir de la dispersión común y de la variabilidad real de cada gen se le asignará a cada uno de ellos una dispersión más propia de este, por lo que para calcular estas estimaciones previamente se deberá haber estimado la dispersión común. Estas estimaciones se obtendrán con la función **estimateTagwiseDisp( )**:

estimateTagwiseDisp(object, prior.df=10, method="grid", ...)

De nuevo en *object* citaremos el objeto al que se le está realizando el estudio, y los demás atributos se dejarán por defecto.

Pruebas

Llegados a este punto se tienen, unos datos con los que se puede trabajar bajo el paquete **edgeR**, pues están bajo la clase *DGEList*, están normalizados, por lo que se pueden comparar entre sí y se ha ajustado una distribución teórica, la distribución Binomial Negativa, para la cuál ya están estimados los parámetros. Por tanto ya se puede prodecer a identificar los genes DE.

El paquete **edgeR** incluye la función *exactTest*, que realiza las pruebas por parejas para la expresión diferencial entre dos grupos.

exactTest(object, pair=1:2, dispersion = "auto", big.count=900, prior.count=0.125)

Solo se nombrará el atributo *object*, donde se volverá a poner nuestro objeto, de clase *DGEList*, ya normalizado y con las estimaciones calculadas.

Con esta función se obtendrán tres elementos:

1. Bajo el dominio *...$table* habrá un data.frame donde se pueden observar por columnas el $log_2 fold - change$ **(logFC)**, los logaritmos en base 2 de la media de los cpm (conteos por millon) de cada gen **(logCPM)** y el p-valor bilateral **(PValue)**.

2. Bajo *...$comparison* se puede ver un vector de caracteres con las condiciones de los dos grupos que se comparan.

3. Bajo *...$genes* un data.frame donde aparecerá la información que se le haya dado, en caso de haberlo hecho, de los genes al crear el objeto de la clase *DGEList*.

Sin embargo, tras calcular las pruebas exactas, se aplicará a estos resultados la función *topTags(·)*, pues con esta función también se tendrán varios elementos, entre ellos una tabla, igual que la obtenida con la función anterior, sin embargo tendrá una columna en la que encontraremos el p-valor ajustado según el método que elijamos, que en nuestro caso será el método de B-H, explicado en la Sección 5.3.1, y en caso de que al crear el objeto *DGEList* se haya incluido alguna información de los genes, también nos aparecerá una columna con dicha información. Es para lo único que serviría incluir información complementaria de los genes, para que cuando se obtenga esta tabla aparezca.

topTags(object, n=10, adjust.method="BH", sort.by="PValue")

Además de lo anterior, esta función permite obtener los resultados ordenados por alguna columna, por defecto lo ordena según la columna *PValue*.

Por otro lado, aunque internamente calcule estos valores para todos los genes solo aparecerán por defecto, los datos referidos a los primeros 10 genes con menor p-valor, si se desea que muestre más genes se tendrá que cambiar este valor por defecto, poniendo el número en el atributo n de la función.

Los demás elementos que proporciona esta función son vectores donde se muestra el método elegido para el cálculo del p-valor ajustado, el tipo de test que se ha realizado y los nombres de los dos grupos que se están comparando.

Realmente estas funciones son muy útiles, pues evitan el tener que calcular esos datos a mano, pero hay veces que es mejor saber y comprobar lo que en realidad está haciendo estas funciones internamente, para saber que calcula realmente lo que queremos. En el caso práctico se harán además, los cálculos manualmente para comprobar que realmente son los datos que se buscan.

6.1.1. Ejemplo práctico

Para este caso práctico nos involucraremos en una particularidad del cáncer de próstata. Los genes estimulados con andrógenos (hormonas masculinas) están implicados en la supervivencia de las células del cáncer de próstata y son un blanco potencial de los tratamientos contra el cáncer. Para el estudio se recogieron tres réplicas de muestras RNA de células de cáncer de próstata después del tratamiento con una hormona androgénica, y también se recogieron cuatro réplicas como "grupo control", a partir de células tratadas con un compuesto inactivo.

Nuestro objetivo será identificar genes diferencialmente expresados en ambos grupos, células tratadas con una hormona androgénica y células tratadas con un compuesto inactivo, es decir, grupo control.

Los datos ya secuenciados y mapeados se pueden encontrar bajo el nombre de **pnas_expression.txt** en http://sites.google.com/site/davismcc/useful-documents, donde están a disposición del público. Sin embargo, las etiquetas de cada muestra se deberán crear manualmente, pues el archivo ya no se encuentra disponible para su descarga. Podremos verlo en la página 47 del manual que estamos utilizando, bajo el nombre *targets*.

Comenzamos cargando el paquete **edgeR**, y los datos descargados:

```
# Cargamos el paquete edgeR:
> biocLite("edgeR")
> library("edgeR")

# Cargamos los datos descargados, previamente deberemos fijar
  # como directorio la carpeta donde tengamos guardado los datos:
> x <- read.delim("pnas_expression.txt", row.names=1,
                  stringsAsFactors=FALSE)

# Construimos el data.frame "targets" con los etiquetas de cada
  # muestra:
> Lane <- c(1,2,2,4,5,6,8)
> Treatment <- c("Control","Control","Control","Control","DHT",
                 "DHT","DHT")
> Label <- c("Con1","Con2","Con3","Con4","DHT1","DHT2","DHT3")

> targets <- data.frame(Lane,Treatment,Label)
> rownames(targets) <- Label
> targets
      Lane Treatment Label
Con1    1   Control  Con1
Con2    2   Control  Con2
Con3    2   Control  Con3
Con4    4   Control  Con4
```

```
  DHT1    5       DHT  DHT1
  DHT2    6       DHT  DHT2
  DHT3    8       DHT  DHT3
```

Clase DGEList

Teniendo los datos cargados ya se pueden estudiar, aunque es mejor esperar a tenerlos bajo la clase *DGEList*, pues se tendrán los mismos datos y el estudio podrá ser aún más completo. Vamos a comprobar las dimensiones de los datos actualmente:

```
> dim(x)
  [1] 37435     8
```

Exiten 37435 filas, es decir genes, y 8 columnas, pero si se observa un poco los datos con la función *head(·)*, se puede ver que solo 7 de esas 8 columnas pertenecen a las muestras, pues la última columna es un dato informativo de los propios genes, el cual se puede utilizar cuando se cree el objeto de la clase *DEGList* a través del atributo *genes* como se había explicado anteriormente, aunque, solo sirve para mantener esa información hasta obtener los resultados.

```
> head(x)
                  lane1 lane2 lane3 lane4 lane5 lane6 lane8  len
  ENSG00000215696     0     0     0     0     0     0     0  330
  ENSG00000215700     0     0     0     0     0     0     0 2370
  ENSG00000215699     0     0     0     0     0     0     0 1842
  ENSG00000215784     0     0     0     0     0     0     0 2393
  ENSG00000212914     0     0     0     0     0     0     0  384
  ENSG00000212042     0     0     0     0     0     0     0   92
```

Lo primero que hay que hacer es crear un objeto de la clase *DEGList*, pues es con la que trabaja este paquete. Se hará con la propia función **DEGList**, teniendo en cuenta seleccionar solo las siete primeras columnas de la matriz de conteos, pues la octava columna era informativa.

```
# Creamos el objeto con la clase DGEList
> y <- DGEList(counts=x[,1:7], group=targets$Treatment)
    # Opcionalmente podemos poner el atributo: genes = x[,8]
    # La columna 8 no es una muestra

> colnames(y) <- targets$Label # Le ponemos nombres a las columnas.
```

Como la última columna de los datos originales es únicamente informativa, no se ha incluido en el análisis.

Los grupos están definidos en la segunda columna de *targets*, con el nombre de *Treatment*, que indican si las muestras están estimuladas con andrógenos o no. Tras

crear nuestro objeto, se le asignan los nombres a las columnas, es decir a las muestras, que están en la tercera columna de *targets*, llamada Label.

Cuando se introdujo los datos vimos las dimensiones de la tabla de datos originales, comprobamos que con el objeto de la clase *DGEList* se mantienen estas.

```
> dim(y)
  [1] 37435     7
```

Como se puede observar, se siguen teniendo 37435 genes y 7 muestras. Además en el objeto **y** se recoge nueva e importante información.

```
> head(y)
  An object of class "DGEList"
  $counts
                  Con1 Con2 Con3 Con4 DHT1 DHT2 DHT3
  ENSG00000215696    0    0    0    0    0    0    0
  ENSG00000215700    0    0    0    0    0    0    0
  ENSG00000215699    0    0    0    0    0    0    0
  ENSG00000215784    0    0    0    0    0    0    0
  ENSG00000212914    0    0    0    0    0    0    0
  ENSG00000212042    0    0    0    0    0    0    0

  $samples
         group lib.size norm.factors
  Con1 Control   978576            1
  Con2 Control  1156844            1
  Con3 Control  1442169            1
  Con4 Control  1485604            1
  DHT1     DHT  1823460            1
  DHT2     DHT  1834335            1
  DHT3     DHT   681743            1
```

En *$counts* se encuentran los conteos brutos, y en *$sample* la información de las muestras, si han sido estimuladas o no, y además se han añadido dos columnas, con el tamaño de las muestras, o número de conteos de cada una de ellas y los factores de normalización cuya función es la de ponderar el tamaño de las muestras. Estos últimos son en realidad la suma de todos los conteos que se han obtenido en cada una de ellas, es decir, se puede comprobar estos valores sumando cada columna de la matriz de conteos. Y po último, los factores de normalización son siempre 1 para todas las muestras. Esto se debe a que la función *DGEList*, por defecto, asigna a cada muestra un factor de normalización igual a 1. Posteriormente, a lo largo del proceso de análisis, lo calcularemos, por eso se ha dejado dejado el valor por defecto, ya que no supone ningún inconveniente.

Filtrado de los genes

Es conveniente filtrar los genes antes de continuar con el análisis. Lo que se hará es eliminar los que tienen muy baja expresión, manteniendo los que tienen un nivel razonable. Ya que el tamaño más pequeño de grupo es de tres (la condición DHT solo tiene tres réplicas), se mantendrán los genes que logran al menos un cpm (conteos por millón) de tres.

Vamos a seleccionar estos genes y luego se comentarán los pasos que se siguen para hacerlo.

```
> keep <- rowSums(cpm(y)>1) >= 3
```

La función *cpm(·)* calcula los conteos por millón de cada muestra, es decir, la suma de cada columna tras aplicar esta función será de un millón. Haciendo una regla de tres es fácil comprobarlo:

```
# Para el gen de la fila 506
> colSums(x)[1:7] # Suma por columnas de la matriz de conteos
    lane1   lane2   lane3   lane4   lane5   lane6   lane8
   978576 1156844 1442169 1485604 1823460 1834335  681743

> x[506,1:7] # Conteos exactos para el gen de la fila 506
                 lane1 lane2 lane3 lane4 lane5 lane6 lane8
  ENSG00000062598   117   170   178   219   434   434   149

> (x[506,1:7]*1000000)/colSums(x)[1:7] # Regla de 3
                     lane1    lane2    lane3    lane4    lane5
  ENSG00000062598 119.5615 146.9515 123.4252 147.4148 238.0091

                     lane6    lane8
  ENSG00000062598  236.598 218.5574

> cpm(y)[506,] # Coinciden los resultados.
      Con1     Con2     Con3     Con4     DHT1     DHT2     DHT3
  119.5615 146.9515 123.4252 147.4148 238.0091 236.5980 218.5574
```

Al imponer la condición de $\mathbf{cpm(y) > 1}$, se crea una matriz con valores lógicos donde aparecerá TRUE si la condición se cumple y FALSE si no se cumple. En general, a los valores TRUE se les asignan unos y a los valores FALSE se les asignan ceros, por lo que al calcular luego la suma por filas con la función *rowSums(·)* lo que se hace realmente es contar cuantos valores positivos tenemos por filas en la matriz de valores lógicos construida anteriormente. Pues bien, tras hacer esto, sólo nos quedamos con las filas, es decir, los genes, en los que esta suma o recuento sea mayor o igual a tres.

Los genes seleccionados están en el vector **keep**, y con este se puede eliminar los que no hayan sido seleccionados del objeto que estamos estudiando.

```
> y <- y[keep,] # Nuevo objeto de estudio, ya filtrado.
> dim(y) # Nueva dimensión
  [1] 16494     7
```

Se ha disminuido considerablemente el número de genes para analizar, exactamente nos quedan 16494 genes, por lo que el análisis y por tanto los resultados y las interpretaciones serán más claras.

La matriz de conteos seguirá estando en condiciones para proceder al análisis, sin embargo, el tamaño de las muestras calculadas anteriormente no se ha modificado con este cambio, y se necesita los valores reales.

```
> y$samples$lib.size # Tamaño de librería antes del filtrado
  [1]  978576 1156844 1442169 1485604 1823460 1834335  681743

> y$samples$lib.size <- colSums(y$counts)
> y$samples$lib.size # Nuevo tamaño de la librería
  [1]  976847 1154746 1439393 1482652 1820628 1831553  680798
```

Normalización

Para seguir con el análisis lo siguiente es estimar los factores de normalización que harán que las muestras sean comparables:

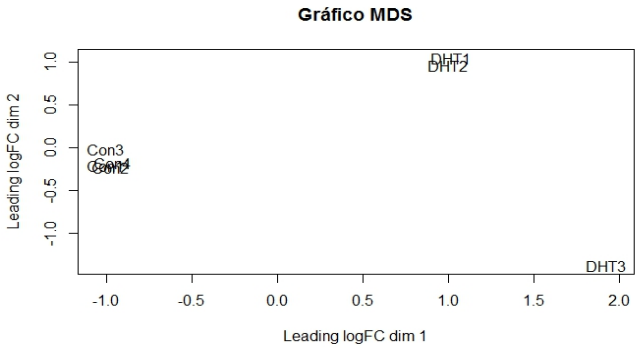
```
> y <- calcNormFactors(y)
> y$samples
        group lib.size norm.factors
Con1 Control   976847    1.0296636
Con2 Control  1154746    1.0372521
Con3 Control  1439393    1.0362662
Con4 Control  1482652    1.0378383
DHT1     DHT  1820628    0.9537095
DHT2     DHT  1831553    0.9525624
DHT3     DHT   680798    0.9583181
```

Tras hacer la normalización la columna donde aparece los factores de normalización cambia, ya que antes estaban los valores por defecto que le asignaba la función **DGEList**.

Exploración de los datos

El gráfico MDS (Multidimensional scaling plot of distances) muestra la relación entre todas las muestras. El análisis que realiza este gráfico es similar a un PCA, es decir, al análisis de componentes principales:

```
> plotMDS(y)
```



Lo que más llama la atención en el gráfico es la separación entre los dos grupos. Las muestras DHT, es decir, las muestras que han sido tratadas con andrógenos, se encuentran en valores positivos del eje X, y las muestras del grupo control en valores negativos, esto supone la mayor diferencia en el gráfico.

Viendo este gráfico se puede ya imaginar la existencia de genes diferencialmente expresados pues existe una clara separación entre los grupos.

Estimación de la dispersión

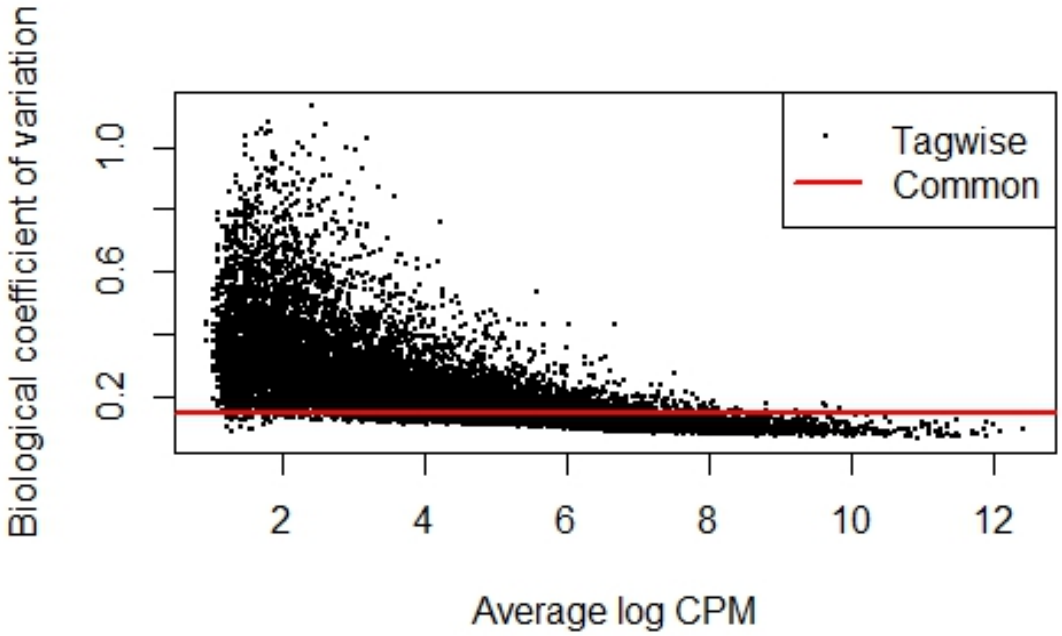
En cuanto a los métodos para estimar las dispersiones, primero se estima la dispersión común, o variabilidad total y luego la dispersión Tagwise, o dispersión gen a gen. Al calcular la dispersión común añadiremos como atributo en la función *verbose* = *TRUE* para que muestre los valores del coeficiente de variación biológico (BCV) y la dispersión común.

```
> y <- estimateCommonDisp(y,verbose=TRUE)
  Disp = 0.02002 , BCV = 0.1415

> y <- estimateTagwiseDisp(y)
```

Ya están estimadas las dispersiones, y como se puede ver en la salida obtenida al estimar la dispersión común, esta es de 0,02 y el coeficiente de variación biológica (BCV) de un 14 %.

Estas dispersiones estimadas se pueden representar con un gráfico BCV, y así se puede comprobar si la dispersión común representa realmente la dispersión existente entre los genes.

```
> plotBCV(y)
```



No es correcto establecer la dispersión común para todos los genes ya que existe mucha diferencia entre ambas dispersiones y por tanto no se puede tratar esta como una buena representación de la dispersión de cada gen.

Pruebas

Una vez calculados los factores de normalización y las estimación de las dispersiones, se puede realizar las pruebas para buscar diferencias entre los dos grupos.

La función que realiza estas pruebas es *exactTest(·)*:

```
> et <- exactTest(y)
> et
  An object of class "DGEExact"
  $table
                        logFC   logCPM      PValue
  ENSG00000124208 -0.43155727 8.725537 0.002740859
  ENSG00000182463  0.69099150 4.691815 0.007133319
  ENSG00000124201 -0.05407528 7.511315 0.667039528
  ENSG00000124207 -0.42270587 5.900093 0.026641325
  ENSG00000125835 -0.24706707 7.098438 0.176066222
  16489 more rows ...

  $comparison
```

```
  [1] "Control" "DHT"

  $genes
  NULL
```

La siguiente función, es para poder obtener también las razones de falsos descubrimientos (FDRs) y ordenar la tabla según la columna"PValue".

```
> top <- topTags(et)
> top
  Comparison of groups:  DHT-Control
                     logFC    logCPM        PValue           FDR
  ENSG00000151503 5.816153  9.714751  0.000000e+00  0.000000e+00
  ENSG00000096060 5.004457  9.948504  0.000000e+00  0.000000e+00
  ENSG00000166451 4.683422  8.844666 5.876578e-250 3.230943e-246
  ENSG00000127954 8.120955  7.212455 1.536671e-229 6.336464e-226
  ENSG00000162772 3.316697  9.742275 1.638026e-216 5.403519e-213
  ENSG00000115648 2.598439 11.474374 7.929928e-180 2.179937e-176
  ENSG00000116133 3.244450  8.790187 7.576826e-175 1.785317e-171
  ENSG00000113594 4.111127  8.047094 4.592178e-161 9.467922e-158
  ENSG00000130066 2.609904  9.987464 1.052458e-154 1.928805e-151
  ENSG00000116285 4.201862  7.359908 3.589473e-149 5.920477e-146
```

Al estar ordenados según la columna "PValue", los primeros elementos serán los que menor p-valor tengan, que coincide, lógicamente, con los menores FDRs, y se ve que existen genes que tienen estos dos datos muy bajos, incluso hay dos genes, *ENSG00000151503* y *ENSG00000096060*, que tienen p-valor y FDR, es decir, p-valor ajustado, cero. Solamente se muestran diez genes de los 16494 genes existentes, en general, será más fácil observar la proporción de DGEs através de gráficos.

Los demás elementos que nos proporciona la función *topTags(·)* son:

```
> top$adjust.method
  [1] "BH"
> top$comparison
  [1] "Control" "DHT"
> top$test
  [1] "exact"
```

Es bueno comprobar cómo trabajan internamente estas funciones, para comprender mejor el método y los resultados que se obtienen. Cada columna de la tabla obtenida con la función *exactTest* nos da la siguiente información:

- Primera columna: Mostraría los datos de la columna 8 de la matriz x, pero no se definió cuando se creó el objeto con la clase *DGEList*, por tanto no aparecerá esta columna, en caso de que hubiera sido definida sería fácil comprobar que aparecen los mismos datos.

- Columna "logFC": el $log_2 Fold - change$.

```
> log2(mean(cpm(y)[rownames(top)[1],5:7])/
            mean(cpm(y)[rownames(top)[1],1:4]))
  [1] 5.825758

> log2(mean(cpm(y)[rownames(top)[2],5:7])/
            mean(cpm(y)[rownames(top)[2],1:4]))
  [1] 5.013882

> et$table[rownames(top)[1:2],1]
  [1] 5.816153 5.004457
```

  Coinciden los valores, aunque algunos decimales varían, pero eso se debe a errores de redondeo.

- Columna "logCPM": Hace el logaritmo en base dos de la media del cpm de y, que recordemos que está filtrado, por lo que no sale lo mismo el cpm(x) que el cpm(y).

```
> log2(mean(cpm(y)[rownames(top)[1],1:7]))
  [1] 9.715967
> et$table[rownames(top)[1],2]
  [1] 9.714751
```

  De nuevo se ve esa variabilidad entre algunos decimales.

- Columna "PValue": En esta columna aparecen los p-valores según una distribución Binomial Negativa con los parámetros estimados anteriormente.

En la tabla de resultados de la función **topTags(·)** se obtienen las mismas columnas que en **exactTest(·)**, añadiendo la columna con la información que aparecía en la octava columna de la matriz de conteos original, que nosotros no hemos considerado. Además incluye una columna nueva:

- Columna FDR: En esta columna lo que hace es calcular el *p.adjust* según el método que se le indique, por defecto es B-H, explicado en la Sección 5.3.1.
  Si se quiere comprobar estos resultados, se puede hacer con la función **p.adjust(·)**, donde se darán los p-valores, que están en la tabla de resultados de *exactTest* y se seleccionará como método B-H.
  Para que salga en el mismo orden que en *top* se deben de coger los p-valores de esta misma tabla, pues ya están ordenados y se podrán comprobar sin estar buscando los genes en la tabla. Sin embargo, en *top* por defecto solo se muestran 10 genes, aunque hace los cálculos con todos los genes. Pero para la función **p.adjust(·)** se necesitan todos los p-valores, así que antes de comprobarlo hay que volver a hacer *top* pero poniendo n=16494, que es la totalidad de los genes existentes.

```
> top2 <- topTags(et,n=16494)
> p.adjust(top2$table[1:16494,3],method="BH")[1:10]
  [1]  0.000000e+00   0.000000e+00  3.230943e-246  6.336464e-226
  [5] 5.403519e-213  2.179937e-176  1.785317e-171  9.467922e-158
  [9] 1.928805e-151  5.920477e-146
```

Aunque los cálculos están hechos para todos los genes, a efectos de comprobación, se ha ordenado que sólo muestre los 10 primeros.

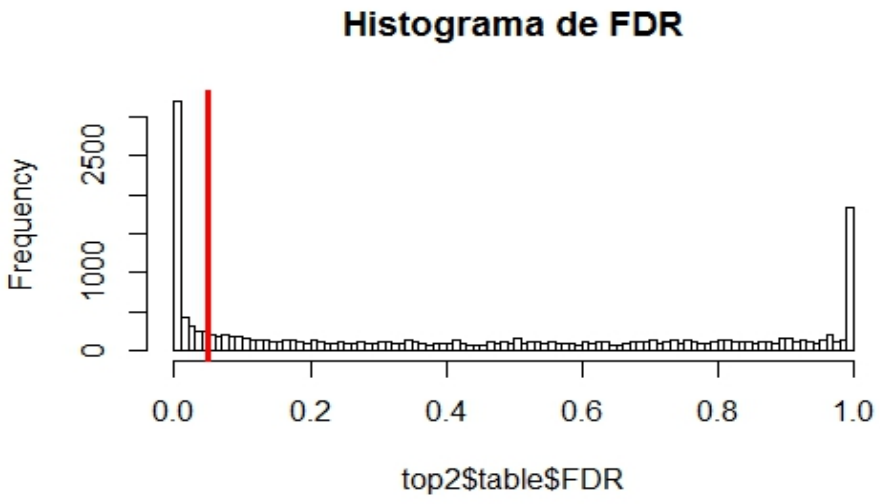
Realmente se obtienen los datos que se perseguían.

Con tantos genes, aunque estén los p-valores ajustados, es muy embarazoso ver cuantos genes están diferencialmente expresados. Por ello es útil realizar algunos gráficos para ver aproximadamente la proporción de genes DE.

Histograma

Se utilizará *top2* para representar todos los genes.

```
> table(top2$table$FDR <0.05)
  FALSE  TRUE
  12059  4435

> table(top2$table$FDR <0.05)/nrow(top2$table)
   FALSE   TRUE
  0.7311 0.2688

> hist(top2$table$FDR, breaks=100, main="Histograma de FDR")
> abline(v=0.05, col="red", lwd=3)
```

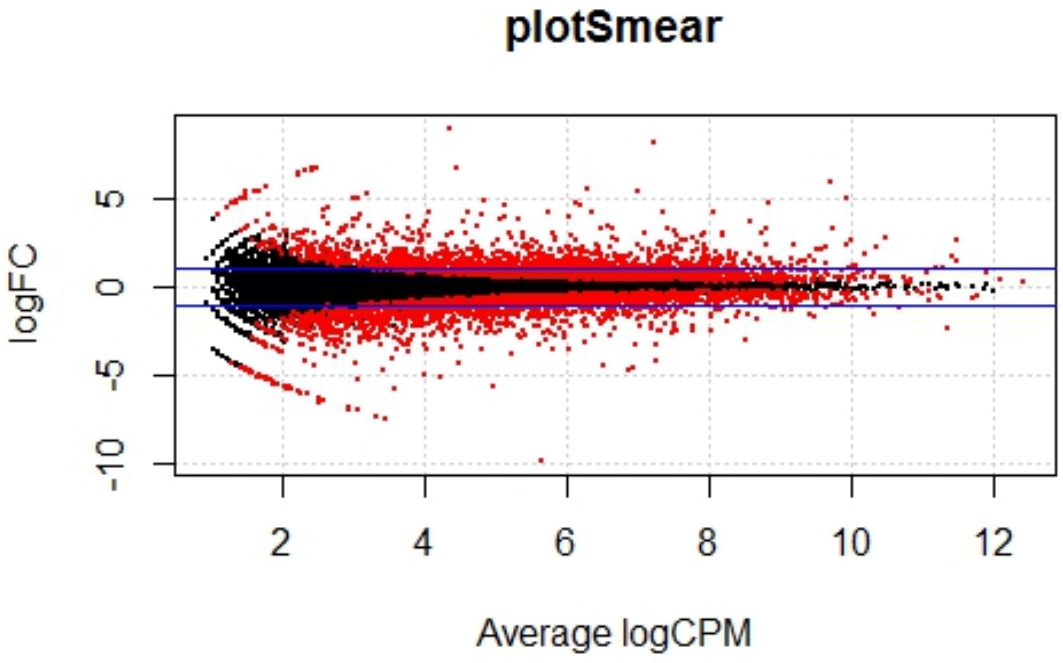


Se ha establecido el corte para $\alpha = 0.05$ para identificar los genes que están diferencialmente expresados. Un 73 % de los genes no se pueden considerar que tengan expresiones distintas en los diferentes grupos, además en el histograma se ve que la mayoría de los datos los encontramos tras el $\alpha = 0.05$.

plotSmear

Este gráfico representa el $log_2 fold - change$ frente a la media del logCPM.

```
> de <- decideTestsDGE(et)
> summary(de)
       [,1]
  -1   2096
   0  12059
   1   2339
> detags <- rownames(y)[as.logical(de)]
> plotSmear(et, de.tags=detags, main="plotSmear")
> abline(h=c(-1,1), col="blue")
```



Primero hay que definir el objeto "de", donde están los genes clasificados atendiendo a los siguentes criterios, primero que el p-valor ajustado, es decir el FDR sea menor de 0.05, y luego que el logFC sea menor que -1 o mayor que 1. Si nos fijamos en la tabla que creamos anteriormente, ya se sabía el número de genes que tenían un p-valor ajustado menor que 0.05, lo que se hace ahora es separar el resto según

el logFC. Tras esto se guarda en cada gen si pertenece a la fila central del *summary* o no, para poder representarlos con distintos colores en un gráfico.
Este gráfico representará el $log_2 fold - change$ frente a la media del logCPM, como vemos en el gráfico.

En el gráfico aparecen de color rojo los genes DE, y los demás en negro, este gráfico tiene una interpretación similar al *MA-Plot* creado con los paquetes **DESeq** y **DESeq2**. Como se podrá observar en los ejemplos prácticos de los próximos paquetes, este gráfico sigue un patrón similar en todos ellos. Existen varios puntos claves que todo gráfico *MA-Plot* va a seguir. Son los siguientes:

1. Van a existir genes DE, representados en color rojo, por encima y por debajo de la línea que delimita los valores que puede tomar el $log_2 fold - change$, esquematizados en la Sección 4.2 en la Figura 4.2, es decir, en Y = 0. Los genes que quedan por encima se deben a que en la condición *Control*, dicho gen ha obtenido más conteos que en la condición *DHT*, y los puntos por debajo lo contrario.

2. Mientras mayores sean los conteos medios, es decir, mientras más a la derecha del gráfico, los genes DE estarán más cerca de la línea delimitadora. Ocurre por la condición logarítmica de $log_2 fold - change$, como se explicó en la Sección 4.2, que mientras mayores sean las medias, aunque se diferencien, el logaritmo se va a diferenciar menos y por consiguiente el límite para determinar que un gen es DE será menor. Pues estamos reduciendo la escala para poder representarlos, pero a la hora de decidir si un gen es DE, tenemos que tener en cuenta la diferencia real.

3. Existe una tendencia a que no existan genes DE a la izquierda del gráfico. Mientras más a la izquierda del gráfico nos encontremos, respecto del eje X, menores serán los conteos que se observa a los genes, y cuando no existen apenas conteos no se pueden mostrar diferencias apenas.

Puede existir cierta variación en este patrón de un paquete a otro, pues cada paquete representa el $log_2 fold - change$ frente a una determinada función de los conteos, la cual puede ser diferente en cada uno de ellos. En este paquete se representa frente a la media del logaritmo de los conteos por millón de cada gen, y en el paquete **DESeq2** por ejemplo, se representa frente a la media de los conteos de cada gen directamente.

Conclusión

En el estudio al que nos hemos referido, este resultado se traduce como que un 26 %, el cual no es un porcentaje bajo, de los genes se expresan diferencialmente al ser estimulados con andrógenos.

6.2. Paquetes DESeq y DESeq2

DESeq creado por Simon Anders, y **DESeq2** por Simon Anders y otros, son paquetes que sirven para analizar datos de conteo obtenidos en ensayos de secuenciación de alto rendimiento como el RNA-Seq y para realizar pruebas de análisis de expresión diferencial. Se puede obtener más información acerca de estos paquetes en la siguientes páginas:

http://bioconductor.org/packages/release/bioc/html/DESeq.html
http://bioconductor.org/packages/release/bioc/html/DESeq2.html

Al igual que con el paquete **edgeR**, se explicarán las funciones más importantes de este paquete y el procedimiento para realizar el análisis de expresión diferencial con el mismo, y luego se realizará un ejemplo práctico para afianzar los conocimientos y explicar las salidas que nos proporcionan las distintas funciones utilizadas. Los manuales que se seguirán en esta sección se pueden encontrar en las siguientes páginas de Bioconductor:

- DESeq: http://bioconductor.org/packages/release/bioc/vignettes/DESeq/inst/doc/DESeq.pdf
- DESeq2: https://bioconductor.org/packages/release/bioc/vignettes/DESeq2/inst/doc/DESeq2.pdf

Realmente **DESeq2** es una mejora del paquete **DESeq**. Entre otras cosas en las que ha mejorado el paquete, lo que hace que estadísticamente sea preferible la nueva versión es que **DESeq2** utiliza estimadores "shrinkage" o estimadores de contracción, para las dispersiones y para el parámetro $log_2 fold - change$, en los cuales la estimación se ve mejorada por incluir en los cálculos algún tipo de información adicional que afecta a los datos. Y por otro lado, como veremos en el ejemplo práctico, **DESeq** toma una postura muy conservadora.

Para nosotros, la gran diferencia de trabajar con un paquete u otro será la cantidad de pasos que se deben de seguir para llegar al análisis de expresión diferencial. En la versión más antigua se realizan los pasos uno a uno, es decir, al igual que con el paquete **edgeR**, se calcularán las estimaciones y demás a través de distintas funciones, sin embargo con el paquete **DESeq2** la mayoría de las funciones se unificarán y esto hará el trabajo más cómodo y rápido. De todas formas, en el ejemplo práctico, se procederá de las dos formas, es decir, utilizando las dos versiones.

Aunque el análisis es similar, los resultados finales no serán del todo iguales, ya que cada versión tiene una forma distinta de trabajar, además de que **DESeq2**, como habíamos dicho, dispone de un material estadístico más potente.

Como **edgeR**, cada paquete trabaja bajo una clase propia, la cual se deberá de crear a partir de los datos para poder comenzar con el análisis.

Clase CountDataSet

Es la clase bajo la que trabaja el paquete **DESeq**, para crearla existe la función **newCountDataSet(·)**, y se requerirá de la matriz de conteos y un factor que muestre la condición que agrupará las muestras o individuos.

newCountDataSet(countData, conditions, sizeFactors = NULL, phenoData = NULL, featureData = NULL)

En el atributo *countData* se dará la matriz con los conteos brutos, es decir la matriz con los conteos originales, cuya estructura será por filas los genes y por columnas las muestras o réplicas y con valores enteros positivos, y en *conditions* se pondrá el factor con la información de cada muestra o individuo que los diferencia. Los demás atributos no tiene sentido definirlos en este estudio.

Clase DESeqDataSet

Por otra parte, **DESeq2** trabaja bajo la clase *DESeqDataSet*, que es similar a la clase *SummarizedExperiment* utilizada para trabajar con el paquete **GenomicRanges**. Para crear un objeto de esta clase está la función **DESeqDataSetFromMatrix (·)**, la cual necesitará de nuevo la matriz de conteos. Si los conteos no están en ese formato existen otras funciones que permiten crear un objeto de esta clase como la propia función **DESeqDataSet(·)**, o la función **DESeqDataSetFromHTSeqCount (·)**. Sin embargo en este estudio se trabajará con matrices de conteos perfectamente preparadas por lo que se podrá utilizar la función nombrada anteriormente.

DESeqDataSetFromMatrix(countData, colData, design, ignoreRank = FALSE, ...)

De nuevo en *countData* irá la matriz de conteos como en la función explicada anteriormente, en *colData* se incluirá la información acerca de las muestras o los individuos, en el que al menos existirá una columna, y cada fila corresponderá a cada columna de la matriz nombrada en *countData*. Dentro de esta información deberá estar la condición que separa a las muestras para el análisis que realizaremos y además se podrá tener más información sobre éstas.

Aunque se puede realizar el análisis utilizando toda la información existente, en este estudio sólo se utilizará la información que indique a qué grupo pertenece cada muestra, por ejemplo, si se habla de individuos tratados con un determinado medicamento, será necesaria la información que muestre si el individuo ha sido o no tratado con dicho medicamento. Sin embargo, es posible utilizar toda la información y realizar el estudio no sólo con una condición, sino también con condiciones adicionales. Por ejemplo, podemos tener la condición de si un individuo ha sido tratado con un determinado tratamiento o no, y también si la muestra ha sido secuenciada de una forma u otra, en caso de tener dicha información. En este caso se deberá de incluir una fórmula en el atributo *design*, escribiendo cada factor bajo estudio unido con un signo de mas (+) tras una tilde($\sim$).

En este ejemplo se refleja el tipo de datos que hay que incluir en el atributo *colData*:

```
#Individuos
> individuos <- c("ind1", "ind2", "ind3", "ind4", "ind5")

#Información sobre estos individuos
> tratamiento <- as.factor(c("No", "Si", "No", "No", "Si"))
> sobrepeso <- as.factor(c("Si", "Si", "No", "Si", "No"))
> altura <- as.factor(c("1.75", "1.62", "1.59", "1.70", "1.68"))

#colData
> colData <- data.frame(tratamiento, sobrepeso, altura)
> rownames(colData) <- individuos

> colData
      tratamiento sobrepeso altura
ind1          No        Si   1.75
ind2          Si        Si   1.62
ind3          No        No   1.59
ind4          No        Si   1.70
ind5          Si        No   1.68
```

Si sólo se utilizara la variable tratamiento para el estudio, en el atributo *design* se deberá de escribir : $\sim$ *tratamiento* , sin embargo si el estudio se hiciera sobre las variables tratamiento y sobrepeso a la vez, se escribirá: $\sim$ *tratamiento+sobrepeso*.

Una vez creado el objeto con la clase *CountDataSet* o *DESeqDataSet* se puede proceder con el análisis de los datos.

Filtrado de los genes

Por la misma razón que en el paquete **edgeR**, donde se ha explicado más detenidamente este paso y el siguiente, y aunque en el manual de estos no lo indique, es conveniente realizar un filtrado previo de los genes, pues existen genes que no tienen conteos en ninguna de las muestras, o que apenas se expresan y lo único que hace es engordar los gráficos y dificultar las interpretaciones cuando de primera mano se sabe que no se obtendrá de ellos genes que se expresen diferencialmente.

Para eliminar estos genes se puede seguir el mismo procedimiento que con el paquete anterior, es decir, manualmente, pues no existe una función especifica que lo realice.

Normalización

Como primer paso necesitamos estimar el valor o razón que lleva a una misma escala los conteos. A este paso se le suele llamar "normalización". Como se había explicado en la Sección 1.4, lo que se pretende es hacer comparables los datos.

Este paquete utiliza el primer método explicado en la Sección 1.4. Para ello primero se estimará los tamaños, o cantidad de recuentos de las muestras, que veremos en R como *size factors vector*.
Bajo el paquete **DESeq** existe la función *estimateSizeFactors(·)* para estimar estos valores:

estimateSizeFactors(object, locfunc = median, geoMeans, controlGenes, normMatrix)

En esta función únicamente se tiene en cuenta el primer atributo, *object*, donde se nombrará el objeto con la clase *CountDataSet*, que previamente se ha creado.

Sin embargo, en **DESeq2**, no será necesario este paso, pues con dos funciones se realiza todos los cálculos intermedios y únicamente habrá que crear el objeto de la clase *DESeqDataSet* y con esas funciones, explicadas posteriormente, se podrá pasar directamente a recoger los resultados del análisis.

Estimación de la dispersión

La estimación de la dispersión de cada gen en ambos paquetes se basará en 3 pasos. En primer lugar, estima un valor de dispersión para cada gen y luego se ajusta una curva a través de esas estimaciones. Por último, el paquete **DESeq**, asigna a cada gen un valor de dispersión, por defecto elige el mayor valor entre la estimación por gen y el valor ajustado. Cabe decir que este paquete toma una postura muy conservadora y por eso mismo asigna el mayor valor posible a cada gen. La dispersión se puede entender como el cuadrado del coeficiente de variación biológica. Cuanta más dispersión o variación biológica tenga un gen, mayor deberá de ser la diferencia entre los conteos de las distintas condiciones tratadas para que ese gen se tome como significativo en la prueba de expresión diferencial.

La función que utilizaremos bajo el paquete **DESeq** es:

estimateDispersions(object, sharingMode = c("maximum", "fit-only", "gene-est-only"), ...)

En *object* se nombra el objeto de la clase *CountDataSet*, previamente normalizado. En caso de querer cambiar el método de asignación del valor estimado de cada gen se indicará en el atributo *sharingMode*, el cual nos ofrece tres opciones. Por defecto nos implementa el método *maximun*, el cual elige la mayor dispersión para cada gen entre el valor ajustado y la dispersión gen a gen. Para que siempre asigne el correspondiente valor ajustado elegiremos la opción *fit-only* y si por el contrario, se pretende asignar a cada gen la dispersión estimada gen a gen citaremos la opción *gene-est-only*.

DESeq2 realiza un proceso similar para asignar las dispersiones a cada gen, pero tiene en cuenta la media de los conteos de cada gen para calcular la dispersión gen a gen. Sin embargo, este paso no será necesario, pues lo realizará internamente al igual que la normalización.

Pruebas

Las pruebas se realizan bajo test Binomial Negativo, y con el paquete **DESeq** se utiliza la función:

$$nbinomTest(object,\ condA,\ condB,\ ...)$$

En *object* se da el objeto que se está estudiando. Los siguiente atributos, *condA* y *condB*, definen el formato del estadístico $log_2 fold - change$, en el primer atributo se debe de incluir el nivel del factor, el cual contiene la condición que agrupa las muestras o individuos, que se quiera dejar como denominador de dicho estadístico y en el segundo el numerador.

Las columnas de *colData*, es decir, las distintas características de los individuos o muestras estudiadas, están expresadas en factores, con sus distintos niveles, por ejemplo, el factor *condition* tiene como niveles *treated* y *untreated*. Por defecto, R nombra los niveles de los factores por orden alfabético, y para el estadístico $log_2 fold - change$, R tomará el último nivel del factor como numerador y el primero como denominador. En el ejemplo del factor condición utilizaría *treated* como denominador por ser el primero quedando como estadístico:

$$log_2 fold - change = log_2 \left(\frac{\mu(untreated)}{\mu(treated)} \right)$$

Si se quisiera cambiar estos niveles se escribirá primero *untreated* y luego *treated* y el $log_2 fold - change$ quedaría así:

$$log_2 fold - change = log_2 \left(\frac{\mu(treated)}{\mu(untreated)} \right)$$

Llegados a este punto con el paquete **DESeq** ya se tendrían los resultados finales, que se representarán en una tabla con distintos datos sobre cada uno de los genes estudiados. En esta tabla se verá la siguiente información:

- **id**: Identificador o nombre de cada gen.
- **baseMean**: Media de los conteos normalizados de cada gen, sin tener en cuenta las condiciones.
- **baseMeanA**: Media de los conteos normalizados de cada gen, referidas a la primera condición nombrada en la función **nbinomTest(·)**.
- **baseMeanB**: Igual que la anterior pero para la segunda condición nombrada en la función.

- **foldChange**:

$$fold - change = \frac{baseMeanB}{baseMeanA} = \frac{\mu_B}{\mu_A}$$

- $log_2 fold - change$: El logaritmo en base dos del estadístico anterior
- **pval**: P-valor para la significancia estadística de este cambio
- **padj**: P-valor adjustado para test multiples mediante el método de Benjamini-Hochberg, el cuál controla la razón de falso descubrimiento (FDR).

Ya se habría acabado el análisis bajo el paquete **DESeq**. Sin embargo para el paquete **DESeq2**, hasta ahora sólo se ha filtrado los genes y creado el objeto de la clase *DESeqDataSet* con la que poder trabajar. El análisis bajo este paquete es más rápido y sencillo.

Antes de realizar el análisis deberemos de modificar el orden de los niveles del factor con la condición que agrupa las muestras o individuos como anteriormente se ha hecho con la función **nbinomTest(·)**. En el paquete **DESeq2** existe una función que hace que directamente se modifique el factor cambiando el orden de los niveles. Esa función es:

relevel(x, ref, ...)

En *x* se incluirá el factor a tratar y en *ref* citaremos el nivel de referencia, que será el que R use como denominador del estadístico $log_2 fold - change$.

Una vez hecho el cambio de los niveles, el análisis se realizará mediante la función **DESeq(·)**, la cual engloba los pasos anteriores realizados mediante las funciones **estimateSizeFactors(·), estimateDispersions(·) y nbinomWaldTest(·)**, que son prácticamente los pasos que se siguen bajo el paquete **DESeq**:

DESeq(object, test = c("Wald", "LRT"), fitType = c("parametric", "local", "mean"), quiet = FALSE, ...)

En *object*, como siempre, irá el objeto de clase *DESeqDataSet*, recordemos que estamos bajo el paquete **DESeq2**, y los demás atributos no los tocaremos. El atributo *quiet*, por defecto, hace que mientras trabaje la función nos vaya indicando los pasos que va siguiendo hasta terminar el análisis.

Desde este nuevo objeto creado ya se pueden observar los estadísticos, y el p-valor que nos indica si aceptar o rechazar que los genes estén diferencialmene expresados, sin embargo no se han calculado los p-valores ajustados por el método B-H por lo que se ejecutará también la función **results(·)**:

results(object, contrast, lfcThreshold = 0, altHypothesis = c("greaterAbs", "lessAbs", "greater", "less"), pAdjustMethod ="BH", ...)

Únicamente se nombrará el objeto bajo estudio y los demás atributos se dejarán por defecto. R utiliza para calcular los p-valores ajustados el método B-H por defecto, por lo que no se cambiará nada. Sin embargo, si se quisiera utilizar otro método habría que definir cualquiera de los siguiente métodos en el atributo *pAdjustMethod*:

"holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none"

Los resultados se pueden ver también en una tabla como la anterior, exceptuando las bases medias de cada nivel del factor y el fold-change, pues estos cálculos intermedios no los da, sino que nos los ofrece ya directamente con el logaritmo en base 2. Además nos proporciona el error estándar del estadístico $log_2 fold - change$ y el estadístico de Wald con el que realiza el contraste, los cuales se explicaron en la Sección 4.1.4.

Una vez que se obtenga la tabla de resultados, podrían observarse en algunos casos **NAs** como resultado, esto se debe a alguna de las siguientes razones:

1. Si para un gen, todas las muestras tienen conteos cero, la columna *baseMean* será cero para ese gen y la estimación del $log_2 fold - change$, el *pval* y el *padj* obtendrán el valor **NA**.
   Esto se debe a que el fold-change resultaría ser $\frac{0}{0} = IND$ y a partir de ahí no sería posible realizar los cálculos. Pero al hacer el filtrado de los genes previamente esto no sucederá.

2. Si un gen contiene un conteo extremo, outlier, el *pval* y el *padj* serán **NA**.

3. Si un gen tiene una baja frecuencia de conteos normalizados, entonces únicamente el *pval* será **NA**.

6.2.1. Ejemplo práctico

El conjunto de datos que utilizaremos se denomina *pasillaGenes*, el cual se presentó en el artículo *Conservación de un mapa regulatorio RNA entre Drosophila y mamíferos* de Brooks et al. [12]. El experimento trata sobre el cultivo de células de la mosca *Drosophila melanogaster*, también conocida como mosca del vinagre o mosca de la fruta. A partir de estos cultivos se investiga la reducción en la expresión de los genes ortólogos para mamífereos NOVA1 y NOVA2, utilizando el RNAi (ARN de interferencia - *interfering RNA*). El paquete de datos proporciona el recuento de lecturas por exón y por gen calculadas para los genes seleccionados a partir de datos de RNA-Seq.

Suponiendo que ya está cargado **Bioconductor**, lo primero será cargar los datos que pueden encontrar en la propia librería *pasilla*:

```
# Cargamos los datos:
> biocLite("pasilla")
> library("pasilla")
> data("pasillaGenes")
> dat <- pasillaGenes #Guardamos los datos en el objeto "dat"
> dat
  CountDataSet (storageMode: environment)
  assayData: 14470 features, 7 samples
    element names: counts
  protocolData: none
  phenoData
    sampleNames: treated1fb treated2fb ... untreated4fb
      (7 total)
    varLabels: sizeFactor condition type
    varMetadata: labelDescription
  featureData: none
  experimentData: use 'experimentData(object)'
    pubMedIds: 20921232
  Annotation:
```

Como se puede ver los datos tienen una clase *CountDataSet*, por lo que se pueden utilizar las funciones explicadas anteriormente para crear el nuevo objeto con la clase *CountDataSet* para trabajar con **DESeq** y el objeto *DESeqDataSet* para analizar dichos datos mediante **DESeq2**.

Teniendo cualquier conjunto de datos podemos observar a través de la función *str(·)* lo que engloba dicho conjunto. Para llegar al elemento que proporciona las informaciones sobre las muestras, en este caso en particular, podemos utilizar la función *pData(·)*. Sin embargo, no para todos los conujuntos de datos podemos utilizarla, pero siempre podremos recurrir a *str(·)* para saber que comando nos lleva a estas informaciones, en el caso de que el conjunto de datos las contenga.

En este conjunto de datos tenemos la siguiente información sobre las muestras:

```
#Si utilizamos la función que llega a dichos datos:
> pData(dat)
              sizeFactor condition        type
  treated1fb          NA   treated single-read
  treated2fb          NA   treated  paired-end
  treated3fb          NA   treated  paired-end
  untreated1fb        NA untreated single-read
  untreated2fb        NA untreated single-read
  untreated3fb        NA untreated  paired-end
  untreated4fb        NA untreated  paired-end
```

Como se puede observar existen tres factores, *sizeFactor* donde se debería de ver la proporción de los recuentos de cada muestra, aunque aparece como desconocido, pero no pasa nada, pues se estimará luego en la normalización, *condition*, donde se expone si la muestra ha sido tratada o no con cultivos, y *type*, en el que se observa si la muestra ha sido secuenciada una o dos veces. Utilizaremos el segundo factor para el estudio, que es el que informa sobre si la muestra ha sido tratada o no con cultivos, aunque se guardará también el tipo de secuenciación para mostrar como funciona el atributo *design* para la función **DESeqDataSetFromMatrix(·)**.

```
> descripcion <- pData(dat)[,c("condition","type")]
> descripcion
              condition        type
  treated1fb     treated single-read
  treated2fb     treated  paired-end
  treated3fb     treated  paired-end
  untreated1fb untreated single-read
  untreated2fb untreated single-read
  untreated3fb untreated  paired-end
  untreated4fb untreated  paired-end
```

Se realiza esta operación porque hay que tener por separado la matriz de conteos y las características de las muestras para poder utilizar la función **DESeqDataSeqFromMatrix** y **newCountDataSet**, si los datos tuvieran la clase *summarizedExperiment*, por ejemplo, no habría que definir cada aspecto por separado.

Llegados a este punto, ya están cargados los datos, y definida la condición que separa las muestras en grupos, lo que queda, para poder crear el objeto en la clase correspondiente para trabajar con cada paquete, es definir la matriz con los conteos brutos.

```
> matrizconteo <- counts(dat)
> head(matrizconteo)
              treated1fb treated2fb treated3fb untreated1fb
  FBgn0000003          0          0          1            0
  FBgn0000008         78         46         43           47
```

```
  FBgn0000014          2          0          0          0
  FBgn0000015          1          0          1          0
  FBgn0000017       3187       1672       1859       2445
  FBgn0000018        369        150        176        288

             untreated2fb untreated3fb untreated4fb
  FBgn0000003            0            0            0
  FBgn0000008           89           53           27
  FBgn0000014            0            1            0
  FBgn0000015            1            1            2
  FBgn0000017         4615         2063         1711
  FBgn0000018          383          135          174
```

La función anterior, *counts(·)*, devuelve una matriz de enteros positivos con los conteos del conjunto de datos proporcionado.

Filtrado de genes

Como se ha explicado en el paquete anterior, **edgeR**, existen numerosos genes con conteos prácticamente nulos, que no servirán para nada a lo largo del estudio, por lo que lo eliminaremos y así tendremos menos genes con los que trabajar. Sin embargo en el paquete anterior había una función, *cpm(·)*, que calculaba los conteos por millón de cada muestra y se utilizaban para eliminar los genes que tenían una baja frecuencia de expresión, una vez se habían puesto, de alguna manera, en una misma escala todas las muestras. Pero en este paquete no existe tal función ni ninguna similar, por lo que existen dos opciones, una sería utilizar esta función aunque sea de un paquete distinto del que se está estudiando, o eliminar simplemente los genes que no se expresan en ninguna muestra, es decir, que siempre tienen conteos nulos. De la segunda forma nos quedaríamos con más genes, pero se seguiría trabajando dentro del paquete bajo estudio por lo que se hará de esa forma.

Como habíamos dicho, no existiría inconveniente alguno en utilizar una función de otro paquete siempre que no modifique de forma errónea los datos, pero como estamos estudiando este paquete, vamos a ver cómo se puede ir trabajando sin la necesidad de utilizar otros, independientemente de los básicos de R, lógicamente.

```
> suma <- rowSums(matrizconteo)
> matriznueva <- matrizconteo[suma != 0,]
> dim(matriznueva)
  [1] 11836     7

> head(matriznueva)
             treated1fb treated2fb treated3fb untreated1fb
  FBgn0000003          0          0          1            0
  FBgn0000008         78         46         43           47
```

```
  FBgn0000014          2          0          0          0
  FBgn0000015          1          0          1          0
  FBgn0000017       3187       1672       1859       2445
  FBgn0000018        369        150        176        288

             untreated2fb untreated3fb untreated4fb
  FBgn0000003            0            0            0
  FBgn0000008           89           53           27
  FBgn0000014            0            1            0
  FBgn0000015            1            1            2
  FBgn0000017         4615         2063         1711
  FBgn0000018          383          135          174
```

Finalmente nos quedamos con 11836 genes de los 14470 que habían al principio. Realmente, al final, vamos a observar los genes que se diferencian a través de las condiciones que le demos, y éstos obviamente no estarán entre ellos, pero a la hora de realizar gráficos y demás se tendrán unas gráficas e interpretaciones más claras, pues quitaremos la representación de 2634 genes.

DESeq

En el paquete **DESeq** el análisis se realiza paso a paso, sin embargo en **DESeq2** a través de las funciones **DESeq(·)** y **results(·)** se puede realizar todo el análisis.

Primero se va a realizar el estudio bajo **DESeq** y para ello se deberá de cargar el paquete con el que se va a trabajar:

```
# Cargamos el paquete DESeq:
> biocLite("DESeq")
> library("DESeq")
```

CountDataSet

Para empezar con el análisis se generará de nuevo un objeto *CountDataSet*, bajo el que trabaja este paquete, donde hay que incluir el factor que agrupa las muestras o individuos, es decir, la condición a estudiar, que en nuestro caso es si las muestras o individuos han sido tratados o no tratados con cultivos.

```
> cds <- newCountDataSet(matriznueva, descripcion[,1])
```

Normalización

Lo siguiente es calcular los factores de normalización que nos daban una especie de razón de escala o proporción de los conteos que recibía cada muestra o individuo.

```
> cds <- estimateSizeFactors(cds)
> sizeFactors(cds)
  treated1fb   treated2fb   treated3fb  untreated1fb  untreated2fb
   1.5116926    0.7843521    0.8958321     1.0499961     1.6585559

  untreated3fb untreated4fb
     0.7117763    0.7837458
```

Estos serían los factores de normalización de cada muestra o individuo, los que realmente son, una estimación del tamaño de las muestras. Si volviéramos atrás, en este ejemplo, se puede comprobar que este factor aparecía como desconocido, pues bien, de esta forma ya estaría completo el data.frame que se obtenía con la función **pData(·)**.

Con estos datos se pueden calcular los conteos normalizados manualmente o simplemente añadiéndole a la función **counts(·)** el atributo *normalized = TRUE*.

```
> head(counts(cds, normalized=TRUE))
                treated1fb treated2fb  treated3fb untreated1fb
  FBgn0000003    0.0000000    0.00000    1.116281      0.00000
  FBgn0000008   51.5977927   58.64713   48.000064     44.76207
  FBgn0000014    1.3230203    0.00000    0.000000      0.00000
  FBgn0000015    0.6615102    0.00000    1.116281      0.00000
  FBgn0000017 2108.2328875 2131.69565 2075.165550   2328.58014
  FBgn0000018  244.0972499  191.24064  196.465378    274.28674

              untreated2fb untreated3fb untreated4fb
  FBgn0000003     0.000000     0.000000     0.000000
  FBgn0000008    53.6611403    74.461598    34.449947
  FBgn0000014     0.000000     1.404936     0.000000
  FBgn0000015     0.6029342    1.404936     2.551848
  FBgn0000017  2782.5411490 2898.382591  2183.105888
  FBgn0000018   230.9237833  189.666335   222.010768
```

Estimación de la dispersión

```
> cds <- estimateDispersions(cds)
```

La función anterior calcula primero la dispersión que se podría definir como gen a gen, luego la común, y por último decide cuál de las dos asignar a cada gen mediante un proceso que se explicará más abajo mediante un gráfico con el que lo veremos más claramente.

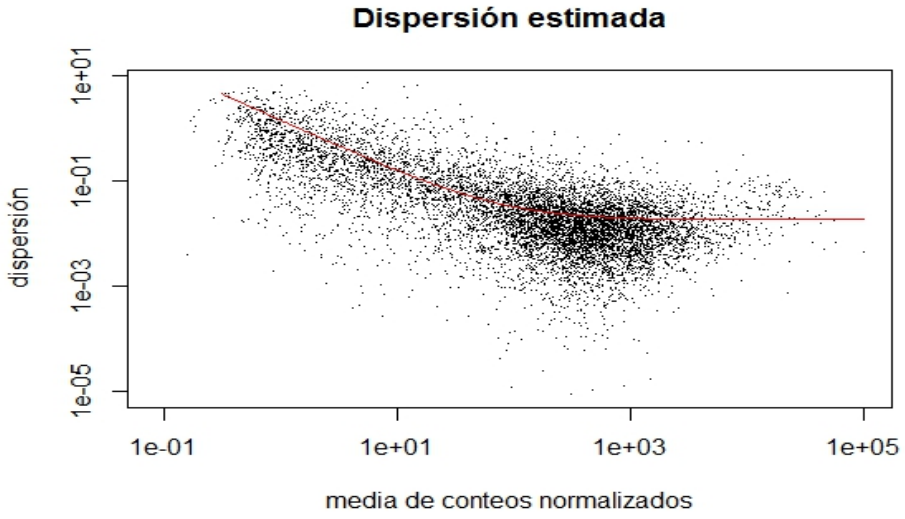
Estas estimaciones se pueden ver a través de la siguiente función:

```
> str(fitInfo(cds))
  List of 5
   $ perGeneDispEsts: num [1:11836] 0.0037 0.048 0.8143 -0.1054 ...
   $ dispFunc       :function (q)
    ..- attr(*, "coefficients")= Named num [1:2] 0.0179 1.3967
    .. ..- attr(*, "names")= chr [1:2] "asymptDisp" "extraPois"
    ..- attr(*, "fitType")= chr "parametric"
   $ fittedDispEsts : num [1:11836] 8.7763 0.0446 3.6018 1.5606 ...
   $ df             : int 5
   $ sharingMode    : chr "maximum"
```

Con el comando *fitInfo(cds)$perGeneDispEsts* se pueden ver las dispersiones estimadas por gen, en *fitInfo(cds)$dispFunc* aparecen los coeficientes de la curva ajustada de las dispersiones estimadas, y por último en *fitInfo(cds)$fittedDispEsts* se ve la dispersión estimada que se ha asignado a cada gen finalmente.

Para representarlas hay que programar previamente una función como esta:

```
> plotDispEsts <- function(cds){
   plot(rowMeans(counts(cds, normalized=TRUE)), fitInfo(cds)
        $perGeneDispEsts, pch = ".", log="xy", ylab="dispersión",
        xlab="media de conteos normalizados",
        main="Dispersión estimada")
   xg = 10^seq(-.5, 5, length.out=300)
   lines(xg, fitInfo(cds)$dispFun(xg), col="red")
 }

> plotDispEsts(cds)
```



Esta función lo que hace es representar la media de los conteos normalizados de cada gen frente a su dispersión por gen y a la dispersión común o curva ajustada a la dispersión de cada gen, estando la primera representada con puntos negros y la segunda con la línea roja.

Es interesante comentar lo que ocurre en el eje Y, donde se observa, que mientras aumenta los conteos normalizados disminuye la dispersión. Los puntos por debajo de la línea roja pueden deberse al bajo número de réplicas de las muestras o individuos estudiados, por eso, **DESeq**, que toma un enfoque conservador, le asigna a los genes que quedan por debajo de la curva el valor ajustado y le mantiene a los demás la dispersión estimada por gen, es decir, le asigna a cada gen la dispersión máxima posible, pues como ya se había nombrado es un enfoque puramente conservador.

Pruebas DESeq

A continuación realizamos el último paso del análisis, que es el test Binomial Negativo sobre los datos, habiendo calculado ya los factores de normalización y estimado la dispersión de cada gen.

Para realizar este test como ya se había comentado, aparte de ofrecer los datos se deben de nombrar por orden los niveles del factor de agrupación de las muestras o individuos. Según el orden establecido se tomará como denominador del estadístico $fold - change$ un nivel o otro. Vamos a utilizar el nivel *untreated*, que define las muestras o individuos que no han sido tratadas como denominador, por eso se nombrará este nivel primero.

```
> res <- nbinomTest(cds,"untreated","treated")
> head(res)
            id     baseMean   baseMeanA    baseMeanB foldChange
  1 FBgn0000003    0.1594687    0.000000    0.3720935        Inf
  2 FBgn0000008   52.2256776   51.833689   52.7483285  1.0176456
  3 FBgn0000014    0.3897080    0.351234    0.4410068  1.2555927
  4 FBgn0000015    0.9053584    1.139929    0.5925969  0.5198540
  5 FBgn0000017 2358.2434078 2548.152442 2105.0313621  0.8261010
  6 FBgn0000018  221.2415562  229.221907  210.6010886  0.9187651

            id    log2FoldChange         pval       padj
  1 FBgn0000003               Inf   0.86914737  1.0000000
  2 FBgn0000008        0.02523529   0.94429078  1.0000000
  3 FBgn0000014        0.32836850   0.84677545  1.0000000
  4 FBgn0000015       -0.94382157   1.00000000  1.0000000
  5 FBgn0000017       -0.27560986   0.06890543  0.6982231
  6 FBgn0000018       -0.12223204   0.62488664  1.0000000
```

Este test ofrece como salida una tabla donde en las filas se pueden ver todos los genes estudiados y en las columnas distintos valores referidos a cada uno de los genes.

En las dos últimas columnas se ven los p-valores, primero el usual y luego el p-valor ajustado para test múltiples de B-H. En algunos genes, como en el primero, se observan *Inf*, esto se debe a que la base media de las muestras que pertenecen a la condición propuesta como denominador del $fold-change$ tiene media de conteos cero, y por tanto resulta infinito.

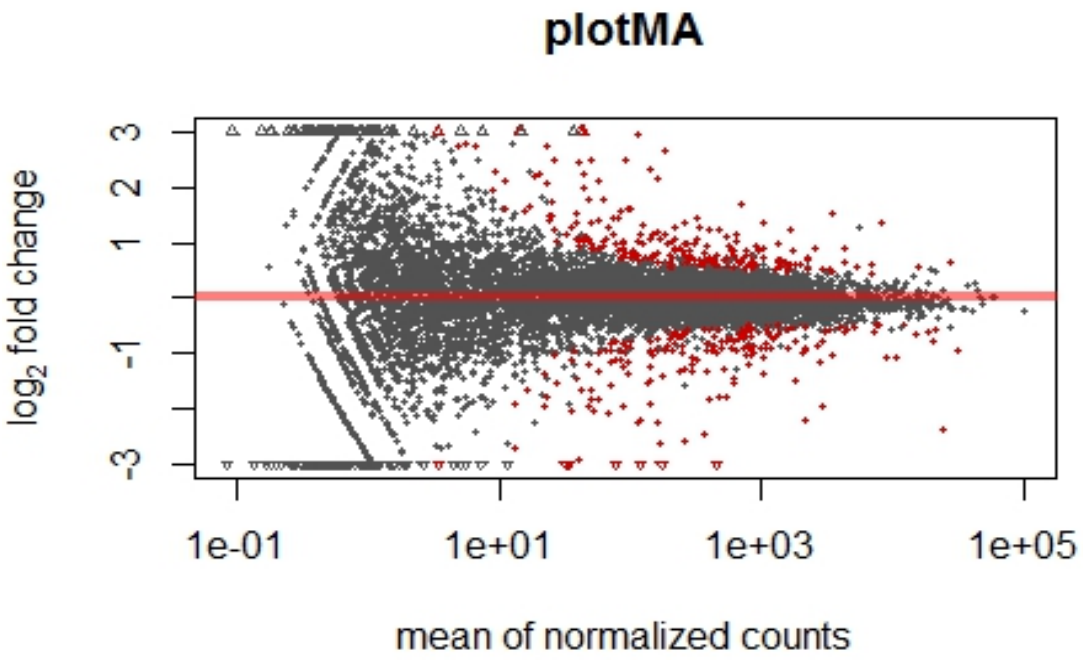
Para ver los resultados más claramente sería conveniente seleccionar únicamente los genes DE, es decir, los genes expresados diferencialmente, para ello marcaremos como límite un $\alpha = 0{,}1$:

```
> resSig <- res[res$padj < 0.1, ]
> head(resSig[order(resSig$pval),])
                id  baseMean   baseMeanA  baseMeanB  foldChange
  8480 FBgn0039155  453.2753   765.46478   37.02275  0.04836637
  2777 FBgn0029167 2165.0445  3258.32360  707.33903  0.21708680
  28   FBgn0000071  180.0102    55.29743  346.29386  6.26238561
  6037 FBgn0035085  366.8279   565.23993  102.27848  0.18094702
  5841 FBgn0034736  118.4074   192.36968   19.79098  0.10287993
  2972 FBgn0029896  257.9027   401.35677   66.63062  0.16601344

                id  log2FoldChange          pval           padj
  8480 FBgn0039155       -4.369852  5.028015e-86   5.951159e-82
  2777 FBgn0029167       -2.203656  6.121511e-37   3.622710e-33
  28   FBgn0000071        2.646712  1.087590e-35   4.290903e-32
  6037 FBgn0035085       -2.466361  2.510743e-35   7.429288e-32
  5841 FBgn0034736       -3.280966  3.641298e-34   8.619680e-31
  2972 FBgn0029896       -2.590628  1.269128e-33   2.503567e-30

> dim(resSig)
  [1] 428   8
```

Por las dimensiones del nuevo objeto creado, dónde solo se han guardado los genes DE, se puede observar que de los 11836 genes que estábamos analizando sólo 428 se expresan diferencialmente según la condición de ser tratados o no por cultivos.

Vamos a ver un gráfico MA-plot donde se puede observar la forma que toman los genes DE, que ya se había visto en el ejemplo práctico del paquete anterior y para el cual se explicó un cierto patrón que sigue este tipo de gráfico.

```
> plotMA(res, main="plotMA")
```



Existe poca cantidad de puntos rojos, que son los que representan a los genes diferencialmente expresados, como ya habíamos visto en el análisis, pues existen pocos genes DE en comparación con los genes estudiados. Si no se hubieran filtrado los genes, simplemente se habría tenido muchos mas puntos sobre la línea roja.

DESeq2

Ahora vamos a volver a realizar el análisis pero con el paquete **DESeq2**. Este hace más fácil el trabajo reduciendo el número de funciones a utilizar.

A la hora de cargar los paquetes se debe de tener en cuenta que para trabajar unicamente con **DESeq** solo deberemos de cargar este y no **DESeq2**, pues existen funciones que trabajan bajo ambos paquetes pero no lo hacen de la misma forma, y al ser **DESeq2** más actual, por decirlo de alguna forma, si se tienen cargados los dos trabajará con las instrucciones de este último. Sin embargo para trabajar con **DESeq2** no importa que este cargado también el anterior.

```
# Cargamos el paquete DESeq2:
> biocLite("DESeq2")
> library("DESeq2")
```

Clase DESeqDataSet

Hasta ahora, antes de empezar con el análisis de los datos bajo el paquete **DESeq**, se había definido la matriz de conteos y el data.frame con información sobre las muestras o individuos, por lo que ya se puede obtener un objeto de la clase *DESeqDataSet*. Se utilizará la matriz de conteos ya filtrada, y para agrupar los individuos, al igual que antes, se hará por la condición de si los individuos son tratados o no con cultivos, por lo que habrá que definir la fórmula en el atributo *design*.

```
# Creamos el objeto con la clase DESeqDataSet
> dds <- DESeqDataSetFromMatrix(countData=matriznueva,
         colData=descripcion, design=~condition)
> dds
  class: DESeqDataSet
  dim: 11836 7
  exptData(0):
  assays(1): counts
  rownames(11836): FBgn0000003 FBgn0000008...FBgn0261574 FBgn0261575
  rowData metadata column names(0):
  colnames(7): treated1fb treated2fb ... untreated3fb untreated4fb
  colData names(2): condition type
```

Ya está listo el objeto con la clase *DESeqDataSet*. Vamos a utilizar ahora la función **relevel(·)** para cambiar el nivel de referencia del factor *condition*.

```
> dds$condition
  [1] treated treated treated untreated untreated untreated untreated
  Levels: treated untreated

> dds$condition <- relevel(dds$condition,"untreated")
> dds$condition
  [1] treated treated treated untreated untreated untreated untreated
  Levels: untreated treated
```

En lo único que afectará el no cambiar los niveles de referencia de los factores será en la interpretación de los resultados. Realmente serán los mismos, sin embargo el $log_2 fold - change$ tendrá signo diferente, debido al cambio del denominador y numerador en la fracción.

Pruebas DESeq2

Para realizar el análisis bajo este paquete, estos son los únicos pasos previos necesarios, aparte de cargar los datos y definir la matriz de conteos y la información de las muestras o individuos, obviamente. Teniendo ya todo esto se puede empezar con el análisis de expresión diferencial.

```
> dds <- DESeq(dds)
  estimating size factors
  estimating dispersions
  gene-wise dispersion estimates
  mean-dispersion relationship
  final dispersion estimates
  fitting model and testing
```

A menos que se defina el atributo *quiet = TRUE*, la función **DESeq**, irá informando de los pasos que va realizando mientras trabaja internamente como se puede ver.

Con esta función se realizan la estimaciones de los tamaños de muestras y de las dispersiones y las pruebas bajo el test Binomial Negativo a través del estadístico de Wald, que son prácticamente los pasos que se han seguido manualmente bajo el paquete **DESeq**. Desde este mismo objeto se observa la tabla final con las medias, los estadísticos y demás, pero no se pueden ver los p-valores ajustado mediante el método B-H, por lo que acudiremos a la función **results(·)**. Por defecto, nos realizará este estudio según el método de B-H, pero como ya se había explicado se puede cambiar el método mediante el atributo *pAdjustMethod*.

Antes de ver la tabla de resultados, vamos a explorar los resultados de esos cálculos intermedios que realiza internamente esta nueva versión del paquete.

```
#Tamaños de muestras normalizados
> sizeFactors(dds)
  treated1fb  treated2fb  treated3fb untreated1fb untreated2fb
   1.5116926   0.7843521   0.8958321    1.0499961    1.6585559

  untreated3fb untreated4fb
     0.7117763    0.7837458

#Dispersiones finales estimadas
> head(dispersions(dds))
  [1] 10.00000000  0.05376325  6.39815207  1.73196457  0.01333305
```

Con los comandos anteriores podemos observar los valores de los cálculos intermedios que esta función realiza. Si miramos atrás se puede comprobar que los tamaños de las muestras normalizados coinciden con los calculados anteriormente con **DESeq**. También se pueden ver las dispersiones asignadas a cada gen, que no coinciden con las anteriores, pues como se había explicado se utilizan estimadores de contracción o estimadores "shrinkage", que hace más certera la información aportada.

Ahora sí, vamos a ver la tabla con los resultados finales.

```
> res <- results(dds)
> head(res)
  log2 fold change (MAP): condition treated vs untreated
  Wald test p-value: condition treated vs untreated
  DataFrame with 6 rows and 6 columns
                  baseMean log2FoldChange      lfcSE       stat
                 <numeric>      <numeric>  <numeric>  <numeric>
  FBgn0000003    0.1594687     0.03455019 0.04614534  0.7487254
  FBgn0000008   52.2256776     0.01971521 0.20925009  0.0942184
  FBgn0000014    0.3897080     0.01181893 0.06215943  0.1901390
  FBgn0000015    0.9053584    -0.04293886 0.10535472 -0.4075647
  FBgn0000017 2358.2434078    -0.25536165 0.11979004 -2.1317436
  FBgn0000018  221.2415562    -0.10384014 0.15577492 -0.6666037

                    pvalue        padj
                 <numeric>   <numeric>
  FBgn0000003   0.45402271          NA
  FBgn0000008   0.92493567   0.9876358
  FBgn0000014   0.84920022          NA
  FBgn0000015   0.68359331          NA
  FBgn0000017   0.03302793   0.2290396
  FBgn0000018   0.50502530   0.8573834
```

En comparación con los p-valores ajustados que se obtuvieron anteriormente, estos son generalmente más bajos. En los casos en los que aparece el valor lógico *NA*, se debe a diferentes circunstancias, no sólo a las veces en las que antes aparecía el valor *Inf*, esta función es más sensible a los casos "extraños" que se presentan a lo largo del proceso de análisis de los datos, las posibles razones ya se han comentado anteriormente.

Vamos a ordenar los genes según el p-valor ajustado.

```
> resOrdered <- res[order(res$padj),]
> head(resOrdered)
  log2 fold change (MAP): condition treated vs untreated
  Wald test p-value: condition treated vs untreated
  DataFrame with 6 rows and 6 columns
              baseMean log2FoldChange     lfcSE      stat
              <numeric>      <numeric> <numeric> <numeric>
  FBgn0039155  453.2753      -3.714214 0.1600580 -23.20543
  FBgn0029167 2165.0445      -2.082793 0.1035963 -20.10491
  FBgn0035085  366.8279      -2.227243 0.1369744 -16.26028
  FBgn0029896  257.9027      -2.206780 0.1586969 -13.90563
  FBgn0034736  118.4074      -2.565002 0.1847628 -13.88268
  FBgn0040091  610.6035      -1.430433 0.1201539 -11.90501
```

```
                      pvalue          padj
                   <numeric>     <numeric>
  FBgn0039155  4.013332e-119 3.069798e-115
  FBgn0029167   6.684462e-90  2.556472e-86
  FBgn0035085   1.888619e-59  4.815349e-56
  FBgn0029896   5.854592e-44  1.119544e-40
  FBgn0034736   8.067450e-44  1.234158e-40
  FBgn0040091   1.114552e-32  1.420868e-29
```

Igual que en la salida que ofrece la función **nbinomTest(·)**, están los genes por filas y sus características por columnas.

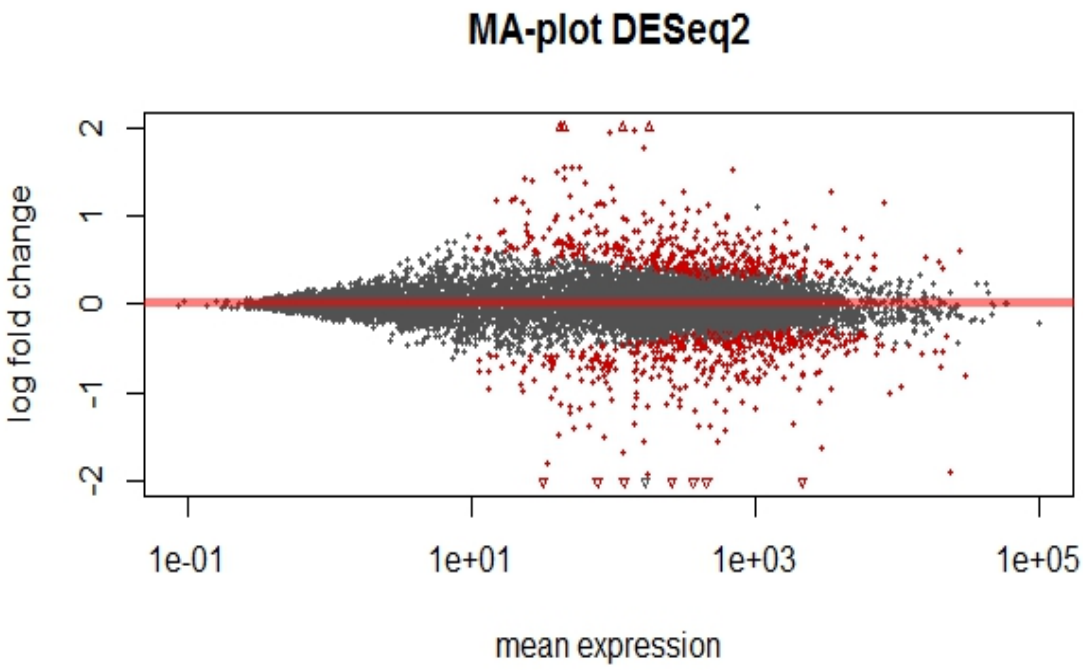
Vamos a ver cuantos genes se pueden considerar realmente como diferencialmente expresados.

```
> xx <-res[order(res$padj,na.last=NA),]
> resSig2 <- xx[xx$padj < 0.1, ]
> dim(resSig2)
[1] 799   6
```

Se han ordenado los genes y seleccionado sólo los que tienen un p-valor ajustado menor que $\alpha = 0{,}1$ y se han obtenido 799 genes que se expresan diferencialmente según si han sido tratados o no con cultivos.

Por último, el nuevo MA-Plot de estos datos sería.

```
> plotMA(res, main="MA-plot DESeq2", ylim=c(-2,2))
```



Como se puede observar es prácticamente el mismo gráfico que el que construimos bajo las instrucciones del paquete **DESeq** exceptuando los puntos más a la izquierda del eje X, donde están más concentrados que en el gráfico anterior. Esto, en gran medida, se debe a que en este gráfico se representa la media de los conteos respecto al $log_2 fold - change$ y en el gráfico anterior se representaba la media de los conteos normalizados.

Conclusión

Como conclusión final del análisis de estos datos, se puede decir que la mayoría de los genes no se ven afectados por ser tratados con ciertos cultivos, pues en ambos estudios hemos obtenido un número muy bajo de genes que se expresan diferencialmente. Tan solo un 6 % en el estudio bajo **DESeq2** y un 3 % en **DESeq**. Como ya se había nombrado, **DESeq** toma un enfoque más conservador que su mejora.

6.3. Comparación entre los paquetes DESeq2 y edgeR.

Aunque hayamos estudiado los paquetes **DESeq** y **DESeq2**, realmente este último es una especie de sucesión del anterior, como ya se había explicado. Por eso, a la hora de elegir uno de estos dos paquetes es conveniente utilizar **DESeq2**. Sin embargo, la duda está entre utilizar éste o el paquete **edgeR**, que por su parte realiza el mismo análisis pero con otros criterios. No se puede decir que paquete de los dos utiliza el mejor criterio a la hora de decidir como estimar las dispersiones, o como normalizar por ejemplo, pues son criterios distintos pero totalmente válidos para el análisis que pretendemos llevar a cabo. Por eso en esta sección trataremos de hacer una comparativa sobre los resultados que nos va dando en cada paso cada paquete, y finalmente compararemos los genes DE que se observan en cada uno de ellos.

Como se verá, las estimaciones que se irán realizando en cada paquete serán totalmente distintas, ya que como se ha explicado, se emplean técnicas diferentes en cada uno de ellos y por tanto los resultados serán distintos.

Para comparar ambos paquetes, nos centraremos en el cáncer de mama en mujeres. El conjunto de datos de RNA-Seq, *breastTCGA*, se han cogido del TCGA (The Cancer Genome Atlas ó según su traducción, El Atlas del Genoma del Cáncer). TCGA es un proyecto destinado a catalogar los datos genómicos de más de 20 tipos de cáncer de diversos institutos de salud. Se puede encontrar más información acerca de este proyecto en la web: *http://cancergenome.nih.gov/*.

Uno de los principales componentes de datos genómicos recogidos por TCGA son los perfiles de expresión génica por RNAseq. El cáncer de mama es el único tipo de cáncer en TCGA donde se recogen los datos de expresión en una gran cantidad de muestras entre las que se pueden encontrar muestras afectadas por dicho tumor y muestras que no presentan dicha enfermedad.

Suponiendo que tenemos cargados los paquetes **DESeq2** y **edgeR**, lo primero que debemos hacer es introducir los datos que vamos a analizar, los cuales están adjuntos en el CD entregado junto al trabajo.

```
## Datos breastTCGA
> load("C:/Users/usuario/Desktop/TFG/RNA-seq/breastTCGA
       /breastTCGA.RData")
> datos <- breastTCGA
> head(datos)
  ExpressionSet (storageMode: lockedEnvironment)
  assayData: 1 features, 757 samples
   element names: exprs
  protocolData: none
  phenoData
   sampleNames: TCGA-A1-A0SB TCGA-A1-A0SD...TCGA-GI-A2C8 (757 total)
```

```
  varLabels: bcr_patient_barcode additional_pharmaceutical_therapy..
    year_of_initial_pathologic_diagnosis (85 total)
  varMetadata: labelDescription
featureData
  featureNames: 1
  fvarLabels: entrezIDs
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

Como se muestra, los datos están bajo la clase *ExpressionSet*, y tenemos disponibles 757 muestras o réplicas. Sin embargo, para estos paquetes es suficiente con tener un mínimo de tres réplicas por condición bajo estudio. Así vamos a utilizar diez muestras en nuestro estudio, cinco de cada grupo o condición. En *varLabels* aparecen una serie de factores que definen a las muestras, entre ellos podemos encontrar el factor *breast_carcinoma_estrogen_receptor_status*, que expresa si la muestra está afectada o no por el tumor, este será nuestro factor que divida las muestras en grupos.

Para comenzar con el análisis vamos a definir la matriz de conteos y el factor que indica a que grupo pertenece cada muestra.

```
> er <- pData(datos)$breast_carcinoma_estrogen_receptor_status[15:24]
> conteos <- exprs(datos)[,15:24]
> dim(conteos)
[1] 20532    10
```

Entre las muestras 15 y 24 hay exactamente 5 muestras de cada condición, que es lo que queremos seleccionar para nuestro estudio, así que escogeremos esas muestras. Tenemos 20532 genes y, evidentemente, 10 muestras de las 757 que había en total.

Para el filtrado de genes, recordemos que en **edgeR**, se utilizaba la función **cpm(·)**. Sin embargo no existe una función parecida para **DESeq2**, por lo que únicamente se eliminaban los genes que no tenían conteos. En la comparación, como es lógico, se deben utilizar los mismos genes, así que realizaremos el filtrado propuesto para **DESeq2**.

```
### Filtrado de genes
> table(rowSums(conteos)==0)
FALSE  TRUE
20105   427
> suma <- rowSums(conteos)
> filtconteos <- conteos[suma != 0,]
> dim(filtconteos)
[1] 20105    10
```

Eliminamos 427 genes de los 20532 existentes.

Una vez filtrados los genes vamos a proceder a realizar el análisis con cada paquete y posteriormente compararemos los resultados.

El análisis bajo **edgeR** sería:

```
## Clase DGEList
> d <- DGEList(counts=filtconteos,group = as.factor(er))

## Normalización
> d <- calcNormFactors(d)

## Estimación de las dispersiones
> d <- estimateCommonDisp(d,verbose=TRUE)
  Disp = 0.50263 , BCV = 0.709

> d <-estimateTagwiseDisp(d)

## Pruebas
> et <- exactTest(d)
> top <- topTags(et, n= Inf)
```

Para ambos paquetes se utiliza por defecto como denominador del $fold-change$ la condición "Negative", así pues, no cambiaremos los niveles de los factores en esta ocasión.

El análisis bajo **DESeq2** sería:

```
### Clase DESeqDataSet
> dds <- DESeqDataSetFromMatrix(countData=filtconteos,
            colData=data.frame(er), design=~er)

### Pruebas
> dds <- DESeq(dds)
  estimating size factors
  estimating dispersions
  gene-wise dispersion estimates
  mean-dispersion relationship
  final dispersion estimates
  fitting model and testing

> res <- results(dds)
```

Ahora que ya está el análisis de los genes bajo los dos paquetes, se pueden comparar los resultados obtenidos en las estimaciones que o bien se realizan paso a paso en el caso de **edgeR**, o bien lo hace internamente R, como en **DESeq2**.

Empezamos por la normalización, que como se explicó, cada paquete la realiza a través de un método distinto, por lo que no es de esperar obtener los mismos valores en estas.

```
#edgeR
> d$samples$norm.factors #edgeR
 [1] 1.1309557 1.0148764 0.9946672 0.9919527 1.0084782
 [6] 0.9514199 0.9608997 0.8801690 0.9854899 1.1041740

#DESeq2
> sizeFactors(dds) #DESeq2
         1         2         3         4         5
 0.7564597 0.9291683 1.1344983 0.7169084 1.2642613
         6         7         8         9        10
 0.7837727 1.0707481 0.7516897 1.2408128 1.4901909
```

Como se puede ver se tienen valores totalmente diferentes del factor de normalización. De hecho, ni siquiera los dos paquetes llaman a la medida que toman para normalizar por igual, uno la llama factores de normalización y otro factores tamaño.

El siguente paso es comparar los valores de las dispersiones estimadas. Al igual que con la normalización, cada paquete realiza este paso mediante un método diferente, por lo que no deben de coincidir los resultados. Sin embargo los pasos que siguen, sí tienen cierta similitud. En ambos paquetes se calcula varios tipos de dispersiones y luego, basándose en estas, y en la media de los conteos además en el caso de **DESeq2**, internamente deciden qué estimación de dispersión asignar a cada gen, que suele ser una función de las dispersiones calculadas, no una de ellas en cuestión.

Vamos a ver los pasos que sigue cada paquete:

```
### edgeR
# Primero calcula una dispersión común
> d$common.dispersion
  [1] 0.5026255

# Luego una dispersión gen a gen a partir de la común
> head(d$tagwise.dispersion)
  [1] 1.1408426 0.4920944 0.6956854 0.4360942 0.2678557
 # Se queda con la que mejor se ajuste de las dos estimaciones

### DESeq2
# Primero calcula una estimación gen a gen
> head(mcols(dds)$dispGeneEs)
  [1] 0.00000001 0.12827200 0.50178000 0.23369489 0.20928070

# Luego através de un ajuste con la media de los conteos
 # estima la dispersión
> head(mcols(dds)$dispersion)
  [1] 10.0000000 0.1834206 0.5171182 0.2662455 0.2491316
```

Vamos ahora a comparar las pruebas, es decir, los p-valores y demás resultados que cada paquete ha calculado para cada gen. Vamos a dar instrucción para que nos muestre los mismos genes para ver si hay alguna similitud en los resultados

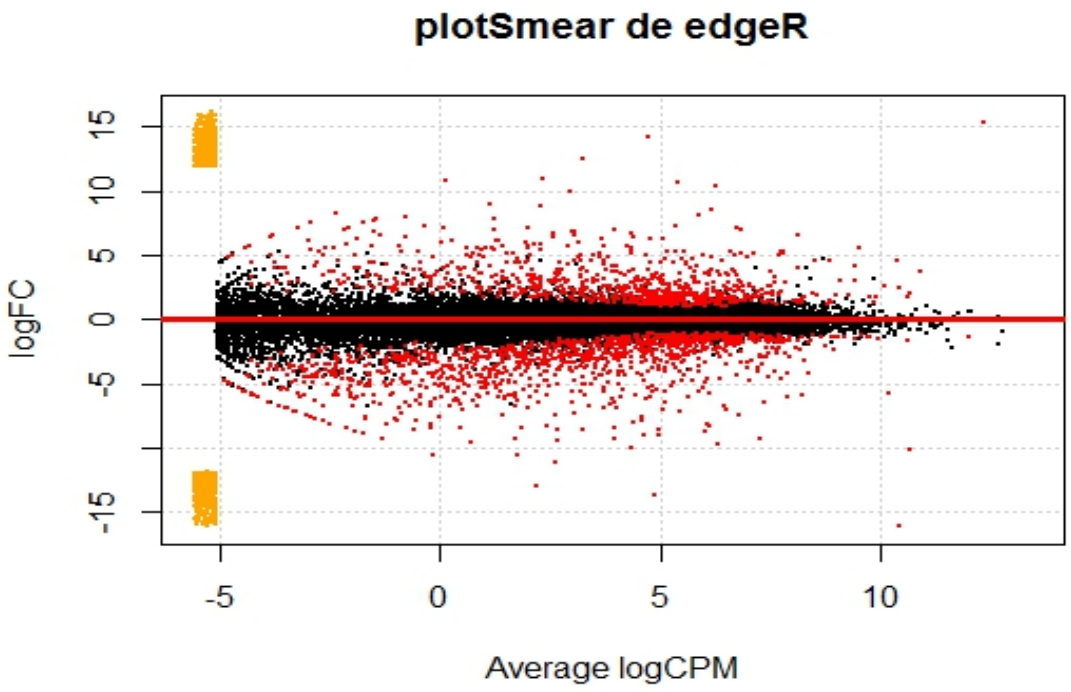
```
### edgeR
> head(top)
  Comparison of groups:  Positive-Negative
           logFC   logCPM       PValue          FDR
  3899   5.132684 6.841526 7.910111e-19 1.590328e-14
  29968 -4.728605 4.620059 8.331744e-17 8.375486e-13
  362   -9.642651 6.313392 4.400588e-16 2.949127e-12
  2568  -5.957511 8.223765 1.252716e-15 6.296464e-12
  22977  5.315483 5.153338 4.647015e-15 1.868565e-11
  26166  7.060679 3.964708 1.069369e-13 3.583278e-10

### DESeq2
> res[rownames(head(top)),]
  log2 fold change (MAP): er Positive vs Negative
  Wald test p-value: er Positive vs Negative
  DataFrame with 6 rows and 6 columns
         baseMean log2FoldChange     lfcSE       stat
        <numeric>      <numeric> <numeric>  <numeric>
  3899  11988.116       4.848521 0.4473020  10.839481
  29968  2603.568      -4.521940 0.4298683 -10.519361
  362    8451.700      -7.668398 0.7754037  -9.889555
  2568  31323.272      -5.387595 0.5870911  -9.176761
  22977  3686.287       4.822263 0.5600524   8.610377
  26166  1601.477       6.040766 0.6818599   8.859249

              pvalue         padj
           <numeric>    <numeric>
  3899  2.237378e-27 3.851423e-23
  29968 7.034923e-26 6.054958e-22
  362   4.620760e-23 1.988544e-19
  2568  4.442508e-20 1.529467e-16
  22977 7.282048e-18 1.253532e-14
  26166 8.055509e-19 1.540750e-15
```

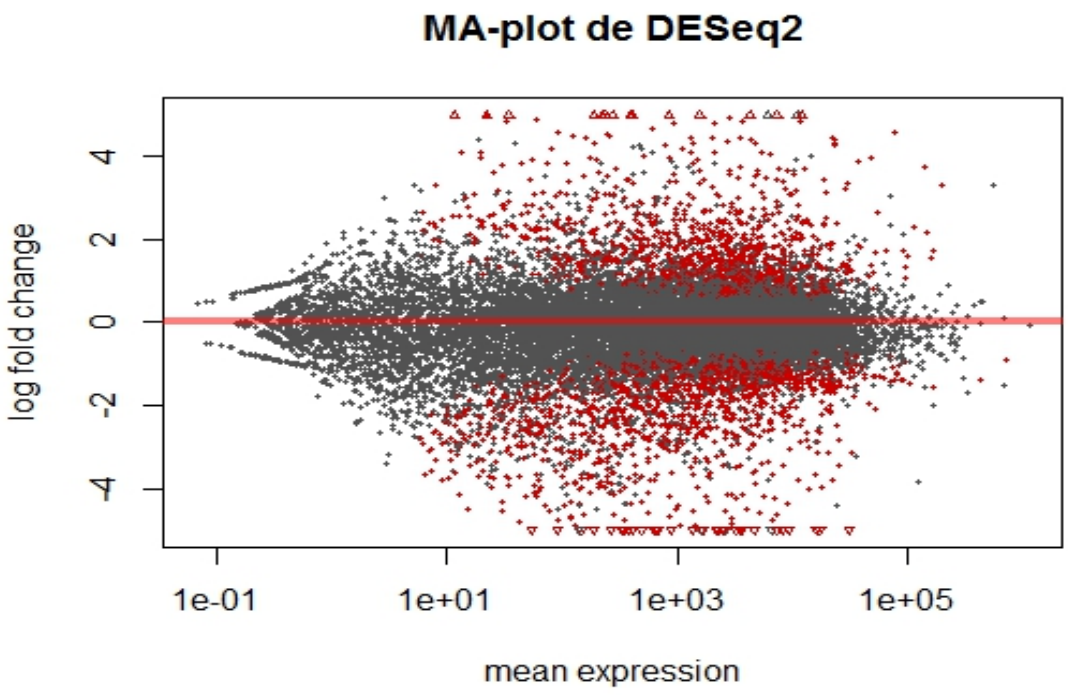
No hay ningún estadístico o parámetro que coincida, ni siquiera el $log_2 fold - change$, esto se debe a que **DEseq2** para calcular este parámetro no utiliza únicamente la información que proporcionan los conteos. Sin embargo, aunque los valores no sean los mismos si están en una cierta escala, por ejemplo, de los genes mostrados en ambos paquete el gen *"362"* es el que menor $log_2 fold - change$ tiene y el que más el gen *"26166"*. Realmente son similares los valores de este parámetro, recalcando el hecho de que bajo el paquete **DESeq2** son algo más bajos. Respecto a los p-valores, luego compararemos con un gráfico de *Venn* los genes DE que tienen en común en ambos paquetes.

Por otro lado, el gráfico **MA-Plot**, representa la media de los conteos normalizados de cada gen frente al $log_2 fold - change$, pero este gráfico no existe en el paquete **edgeR** como tal, sin embargo si existe el gráfico **plotSmear**, que representa algo parecido como es el logaritmo de las medias de los conteos por millón, que recordemos que se calculaban mediante la función **cpm(·)**, frente al $log_2 fold - change$. Por lo que compararemos estos dos gráficos, ya que de algún modo se está representando el parámetro nombrado frente a una función de los conteos. Pero se deberá tener en cuenta que existen distintos valores representados en el eje X.

Por defecto, en el paquete **DESeq2**, utilizan un nivel de significación $\alpha = 0{,}1$, y bajo **edgeR** se utiliza $\alpha = 0{,}05$, por lo que para representarlos tendremos que cambiar alguno de los dos. Utilizaremos un $\alpha = 0{,}1$, por lo que tendremos que definirlo para el gráfico de **edgeR** correspondiente.

```
### edgeR
> de <- decideTestsDGE(et, p.value=0.1)
> detags <- rownames(d)[as.logical(de)]
> plotSmear(et, de.tags=detags, main="plotSmear de edgeR")
> abline(h=0, col="red", lwd=3)

### DESeq2
> plotMA(res, main="MA-plot DESeq2", ylim=c(-5,5))
```





Como se puede observar en los gráficos, aunque presente cierta variabilidad debido a lo comentado anteriormente, vemos ciertas similitudes respecto a la localización de los genes DE. La gran parte de los genes se encuentran entre los valores -5 y 5 del eje Y donde esta representado en ambos gráficos el logFC. Que recordemos que aunque no nos dieran los mismos valores, si eran muy parecidos.

Por último vamos a ordenar los genes según el p-valor adjustado que han obtenido, y nos quedaremos únicamente con los que han sido señalados como genes DE de cada paquete para poder hacer una puesta en común de ellos. Para clasificarlos se va a seleccionar un $\alpha = 0{,}1$, al igual que para los gráficos.

```
#### edgeR
> topSig <- top[top$table$FDR < 0.1, ]
> dim(topSig)
  [1] 1930    4

> genesDEedgeR <- rownames(topSig)

#### DESeq2
# Ordenar por p-valores
> resOrdered <- res[order(res$padj),]

# Solo genes DE
> xx <-res[order(res$padj,na.last=NA),]
> resSig2 <- xx[xx$padj < 0.1, ]
```

```
> dim(resSig2)
  [1] 2689    6

> genesDEDESeq2 <- rownames(resSig2)
```

Bajo el paquete **edgeR** se han obtenido 1930 genes DE y bajo **DESeq2** 2689, por lo que existe una clara diferencia en el número de genes DE obtenidos por cada paquete. Obviamente se debe al procedimiento de cálculo de las estimaciones anteriores. Pero lo más importante ahora es comprobar cuantos genes DE están en común en ambos paquetes. Un paquete puede tomar un enfoque más "permisivo" a la hora de decidir si un gen es DE, pero entonces este también deberá de declarar los genes, o la gran mayoría, de los que un paquete menos permisivo señale como DE.

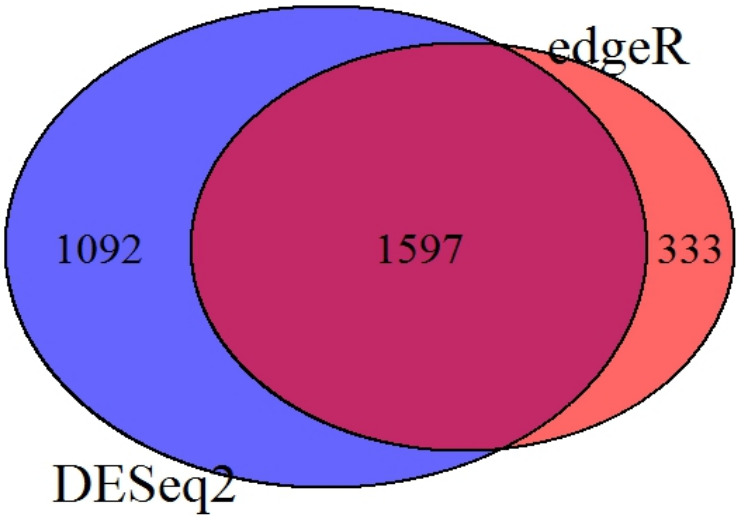
Vamos a ver cuantos genes DE en común existen.

```
> genesDEcomunes <- intersect(genesDEedgeR,genesDEDESeq2)
> head(genesDEcomunes)
  [1] "3899"  "29968" "362"   "2568"  "22977" "26166"

> str(genesDEcomunes)
  chr [1:1597] "3899" "29968" "362" "2568" "22977" ...
```

Existen 1597 genes DE en común en ambos análisis. Si recordamos, con **edgeR** se tenían 1930 genes DE, por lo que la mayoría de estos genes también son declarados como DE en el otro paquete. De esta forma, comprobamos que realmente los dos paquetes trabajan correctamente y que el paquete **DESeq2** es más permisivo que **edgeR**.

Para terminar con la comparación vamos a realizar un gráfico de *Venn* para ver la cantidad de genes DE en común.

```
> install.packages("VennDiagram")
> library(VennDiagram)
> grid.newpage() #Para limpiar la ventana gráfica
> plot2 <- draw.pairwise.venn(2689,1930,1597,c("DESeq2","edgeR"),
                              cat.pos= c(-150,30), fill= c("blue","red"),
                              alpha=0.6,cex=c(2,2,2), cat.cex=c(2.5,2.5))
```



Vemos en la intersección de los dos conjunto los 1597 genes que resultan DE en ambos paquetes. Como podemos ver casi todo el conjunto de genes que resultan DE en **edgeR** también lo son en **DESeq2** como ya habíamos visto.

Conclusión

En este ejemplo, donde se analizaban los genes que eran diferencialmente expresados en muestras afectadas por cáncer de mama o muestras no afectadas por dicho tumor, hemos obtenido que un 13,38 % de los genes son diferencialmente expresados según el estudio bajo el paquete **DESeq2**, en comparación con el 9,60 % que se obtienen a través de **edgeR**. Por lo que se puede considerar que **edgeR** toma un enfoque más restrictivo que **DESeq2**.

Por otro lado, aunque ambos paquetes son totalmente válidos para analizar datos de conteos procedentes de RNA-Seq, si hubiera que elegir uno de ellos, es preferible utilizar DESeq2, pues éste emplea una regresión local de las dispersiones en los datos que le proporciona mayor flexibilidad ante varianzas inestables, puede verse esta demostración en Robles et al. [14]. Además Robles et al. señalan que este paquete utiliza unos estimadores muy potentes.

Bibliografía

[1] J.R. González. Course on 'omic' data analysis with R. April 2014.

[2] A. Mortazavi, B. Williams, K. McCue, L. Schaeffer, and B. Wold. Mapping and quantifying mammalian transcriptomes by rna-seq, *Nat Methods*, 5, 2008.

[3] MD. Robinson and A. Oshlack. A scaling normalization method for differential expression analysis of rna-seq data. *Genome Biol*, 11, 2010.

[4] J.W. Hardin, J.M. Hilbe. *Generalized Linear Models and Extensions.* Stata Press, 2 edition, 2007.

[5] J.M. Hilbe. *Modeling Count Data.* Cambridge University Press, 1 edition, 2014.

[6] A.S. Foulkes (2009). *Applied Statistical Genetics with R For Population-based Association Studies.* Springer-Verlag, 2009.

[7] Y. Benjamini, and Y. Hochberg (1995). *Controlling the False Discovery Rate: a Practical an Powerful Approach to Multiple Testing.* Journal of the Royal Statistical Society B, 57, 289-300.

[8] R Core Team.*R: A Language and Environment for Statistical Computing.* **URL: http://www.R-project.org/**.

[9] R.C. Gentleman, V.J. y Care, D.M. Bates, and others. *Bioconductor: Open software development for computational biology and bioinformatics.* Genome Biology, 5:R80, 2004. **URL http://genomebiology.com/2004/5/10/R80**.

[10] MD. Robinson, DJ. McCarthy and GK. Smyth (2010). *edgeR: a Bioconductor package for differential expression analysis of digital gene expression data.* Bioinformatics 26, 139-140.

[11] Michael I Love, Wolfgang Huber and Simon Anders (2014): *Moderated estimation of fold change and dispersion for RNA-Seq data with DESeq2.* Genome Biology, 15. **URL http://dx.doi.org/10.1186/s13059-014-0550-8**

[12] A.N. Brooks, L. Yang, M.O. Duff, K.D. Hansen, J.W. Park, S. Dudoit, S.E. Brenner, and B.R. Graveley. Conservation of an rna regulatory map between drosophila and mammals. *Genome Research*, 21, 2011.

[13] Simon Anders and Wolfgang Huber (2010). Differential expression analysis for sequence count data. *Genome Biology*, 11:R106. **URL: http://genomebiology.com/2010/11/10/R106/**.

[14] J.A. Robles, S.E. Qureshi, S.J. Stephen, S.R. Wilson, C.J. Burden, et al. (2012). *Efficient experimental design and analysis strategies for the detection of differential expression using RNA-Sequencing*. BMC Genomics 13: 484. doi: 10.1186/1471-2164-13-484