Guillermo Ayala Universidad de Valencia

Bioinformática Estadística

Análisis estadístico de datos ómicos

Guillermo Ayala
Guillermo.Ayala@uv.es

Índice general

Prólogo

Un investigador que publique un artículo al mes durante un año tendrá al final de ese año un total de 12 publicaciones. Después de ese año maravilloso de publicaciones, el siguiente es tan bueno como el anterior. Y sigue publicando un trabajo al mes durante este segundo año. Ya reune 24 publicaciones en dos años agotadores. Y esto mismo lo hace durante 10 años.[1] Al final de esta decada prodigiosa tendrá 120 publicaciones. Ya ha justificado su vida como investigador. Pero no, incansable sigue otros diez años y alcanza al cabo de 20 años los 240 publicaciones. Y diez años más. Exhausto él o ella (y cuantos hayan intentado leer todas estas publicaciones) ya ha llegado a las 360 publicaciones. Hay una enorme cantidad de investigadores que superan esta cantidad. La superan ampliamente. Por amor de Dios, no más. Es seguro (con probabilidad uno) que han participado activamente en todas estas publicaciones. Pero el resto de los humanos no tienen la culpa. Hay numerosos estudios sobre el crecimiento del número de publicaciones. Sin entrar en precisiones innecesarias, en el momento actual, se tarda unos nueve años en doblar el número de publicaciones.[2]

Y debiera de ser posiblemente candidato a algún premio importante.

[1] Asumimos que no es un superhéroe con algún superpoder producido por una pequeña araña radioactiva o algún rayo cósmico por salir sin paraguas a la calle.

[2] Un optimista diría que doblamos el conocimiento que la humanidad ha logrado atesorar cada nueve años.

Evidentemente el planeta no tiene un problema de superpoblación de personas. Es de publicaciones científicas.

[3] Con frecuencia leemos cómo muchas publicaciones han de retirarse[1] Nunca es culpa de nadie. Nunca es cierto que los autores habían apostado por lo rápido. Por lo fácil. Por no controlar el trabajo. Por no limitarse a copiar el tratamiento de datos de otros sin preocuparse si era adecuado en su caso. O controlar el trabajo del precario, mal pagado y sufrido becario/a que echa demasiadas horas y todo es nuevo para él/ella. Necesitaban algunas decenas (centenares) de publicaciones más para que las generaciones futuras los/las recuerden. Y mucho que los recordarán. Yo sé que los recordarán.

[3] 27/09/2019

En campos como las Matemáticas o la Estadística el producto está a la vista. Un teorema lleva su prueba y si hay un fallo puede ser visto y corregido por la comunidad de expertos en ese tema en algún momento posterior. Todo está a la vista. Sin embargo, en campos como la Biología o la Medicina las cosas no son así. No es tan fácil comprobar la reproducibilidad de los resultados. Es muy recomendable leer [11]. Reproduce los resultados de una encuesta a 1576 investigadores. La proporción de trabajos que no se pueden reproducir es enorme. Los factores que se comentan como importantes son los siguientes de mayor a menor frecuencia.

[1] https://www.abc.es/sociedad/abci-retirada-ultima-investigacion-reciente-premio-nobel-202001050137_noticia.html, https://elpais.com/elpais/2019/01/27/ciencia/1548629779_450088.html, https://elpais.com/elpais/2019/10/14/ciencia/1571052466_871787.html.

1. No incluir una descripción completa del trabajo.
2. Presión por publicar.
3. Trabajos sin un estudio de potencia previo y un análisis pobre de los resultados.
4. Insuficiente replicación en el laboratorio original.
5. Insuficiente supervisión.
6. La metodología y el código informático utilizado no disponible.
7. Un diseño experimental pobre o inadecuado.
8. Los datos originales no disponibles.
9. Fraude.
10. Una revisión por parte de la revista insuficiente.

Con el análisis estadístico de los datos tienen que ver los puntos 3, 6, 7 y 8. Hay que hacer bien las cosas. Y desde el principio.

En muchas ocasiones biólogos o médicos o biotecnólogos o bioquímicos o ... me han indicado que querían aprender más de Probabilidad y Estadística. Normalmente después de que les has analizado unos datos para una tesis o un artículo. Al principio, con interés, les recomiendas algún texto esperando que les ayude. El final de la historia suele ser el mismo. Que no leen nunca el libro. Aprender Estadística es muy fácil. Se coge un libro sencillo. El libro lleva una página que pone el número uno. Se lee esa página. Se le da la vuelta y se lee la siguiente numerada con el 2. Y así sucesivamente. Y no hay otra.

Es una opinión poco compartida.

Estas notas tratan de la aplicación de procedimientos estadísticos al *análisis de datos de alto rendimiento*. Bonito el nombre pero: ¿qué son datos de alto rendimiento? Datos que rompen lo que tradicionalmente era un prerrequisito en Estadística (multivariante). Muchos procedimientos estadísticos empiezan indicando que el número de observaciones, n, ha de ser mayor que el número de variables por observación, p. Actualmente es frecuente (mejor habitual) que los datos no los recoja un experimentador con un lápiz y un papel y luego los introduzca con mucho trabajo en una hoja de cálculo. Lo hacen disposivos electrónicos conectados con ordenadores. Por eso los datos tienen dimensiones p que marean: miles de variables frente a decenas (con suerte algo más de un centenar) de observaciones o muestras. ¿Y qué hacemos para analizar esto? Lo que se pueda.[4] De esto van estas notas, *de lo que se pueda*. Son unas notas en progreso. Se van añadiendo ideas, técnicas, software y se ve cómo incorporar estos análisis en nuestro trabajo. También[5] voy incorporando nuevos tipos de información. De momento, analizamos datos de expresión de gen (o expresión génica) sabiendo que (la mayor parte de) lo que hacemos es aplicable en otros contextos. El tema § 4 está dedicado a este tipo de datos, sus características y técnicas de preprocesado. Es el tema específico de esta información. Si se sustituye este capítulo por otro dedicado a otra técnica de adquisición de información casi todo lo que sigue es aplicable.

[4] Uno no está obligado más que a hacer las cosas lo mejor que pueda y no más.

[5] En la medida en que me voy encontrando con nuevos tipos de datos que entienda.

Sobre la parte biológica. El objetivo es que estas notas ayuden a quien lee y no que demuestren que sabe el que las escribe. [6] En lo biológico hay imprecisión por ignorancia, en lo probabilístico y estadístico hay imprecisión porque pretenden ser unas notas para un público interesado en estos temas pero con formación en Biología, Bioquímica o Medicina fundamentalmente. Hay que tener interés en la Estadística y la Informática en cualquier caso para poder seguirlas. En la parte VII se incluye un repaso de procedimientos estadísticos de modo que podamos referenciarlos en el resto de temas según se utilizan. En § 2 se introduce lo básico del uso de R y Bioconductor. Todo en el curso está hecho con este lenguaje y se aprende según se utiliza. No es el énfasis fundamentalmente del curso pero es necesario explicar y usar R continuamente. Y así lo hacemos.

[6] Sobre esta cuestión yo diría que que el autor se apaña como puede (que no es mucho) con los conceptos biológicos.

Sobre R/Bioconductor. Estas notas utilizan sistemáticamente R/-Bioconductor. Y ningún otro software. [7]

Cuando empecé con esto de la Probabilidad y la Estadística uno leía libros y artículos, entendías aquello y luego intentabas descifrar cómo aplicaba (mejor implementaba) estas técnicas el software comercial (en mi caso *SPSS*). Lo primero era bonito. Los autores intentan que les entiendas porque tratan de transmitir ideas. Sin embargo, el software comercial no piensa así (y esto no es necesariamente malo). El software comercial intenta dar un producto bueno y fácilmente utilizable para llegar a un máximo de usuarios. Solamente suelen considerar temas sobre los que hay mucho interés y muchos (potenciales) usuarios. Y esto es correcto. No es la opción elegida en estas notas. Hemos elegido trabajar con software libre. Tanto R como Bioconductor son el resultado de un gran trabajo coordinado de muchas personas. Algunos son proceden del mundo académico. En otros ocasiones proceden de empresas para las cuales les resulta interesante que se disponga de software que permita utilizar su hardware.

[7] En [**Bioconductor2015**] podemos ver un trabajo interesante sobre las prestaciones de Bioconductor.

En lo que sigue hay distintos niveles de lectura en un mismo texto. Es un material que se utiliza en dos cursos claramente diferenciados.

TAMI Uno es una breve introducción en 20 horas lectivas en tercer curso del grado de Bioctecnología de la Universidad de Valencia. En este curso se utilizan las herramientas más básicas, la opción simple y rápida. Se pretende ver cosas interesantes y mantener (en lo que se pueda) la programación con R/Bioconductor simple.

Bioinformática Estadística El segundo curso es un módulo en Bioinformática Estadística en el master de Bioinformática de la Universidad de Valencia. Utilizamos todo el material.

Sobre la investigación reproducible. Es ingente la cantidad de publicaciones científicas que llevan tratamientos estadísticos. Los investigadores suelen saber qué quieren estudiar. Diseñan un experimento y observan datos. Lo que se hace después no lo suelen conocer. No suelen conocer las técnicas que han utilizado.[8] Citan los procedimientos estadísticos y no indican el software utilizado si está disponible en la red o, en el caso de que sea propio de los autores, dónde se puede conseguir.

[8] Y eso es un punto que no tiene solución.

Sin duda alguna el control de la calidad de los tratamientos estadísticos descansa en que cada lector de una publicación científica

tenga a su disposición el artículo (que básicamente es la explicación de lo que se ha hecho) así como los datos [9] y **todo** el código necesario para reproducir **todo** el tratamiento estadístico realizado con los datos. Se repite todo porque en muchas ocasiones lo que se ha descartado puede ser tan interesante como lo que se ha publicado. Este es el conocido sesgo a publicar los resultados significativos descartando los *no* significativos. Sin esto, no se puede realizar un control adecuado de un tratamiento estadístico de datos de alto rendimiento (de hecho, de ningún tipo de datos). Esto nos lleva a los conceptos de programación literaria o comentada (literate programming) propuesto por Donald K. Knuth[10] y, de un modo más genérico, a la investigación reproducible [11]. R/Bioconductor incorpora muchas herramientas para realizar investigación reproducible.[12] En particular, este texto está realizado utilizando [152, knitr]. Todos los datos que se utilizan están disponibles en bases de datos públicas como (sobre todo) GEO o ArrayExpress. No de todos los datos que se utilizados tenemos los datos sin procesado previo. Los usamos por haber sido analizados en otros textos o en ejemplos de R/Bioconductor o, simplemente, porque son bonitos de estudiar.

Es de destacar que recientemente Nature Genetics ha refrendado el uso de Bioconductor.

Sobre la generación simple de informes. Hemos de poder realizar un análisis de datos y generar un informe de un modo sencillo. Esto excluye el uso de herramientas como Word, Excel o similares. En este texto utilizaremos la opción R/Markdown.

Cómo usar el texto. Se intenta explicar Estadística aplicada a la Bioinformática. Por ello es un texto que combina ambas cosas. En ocasiones se utiliza R/Bioconductor como herramienta pedagógica para ilustrar un concepto. En otras casi hacemos de manual técnico para usar un paquete de R. En principio, la idea es que vamos leyendo y ejecutando el código que se inserta. Lo lógico es tener R funcionando y, con un copiar y pegar, podemos ir ejecutando el código.[13]

¿Bioconductor es la única opción? Por supuesto que no. Y tampoco tengo muy claro que sea la mejor. Indudablemente es una de las mejores. En estas notas el enfasis está en el análisis de datos y, en particular, en el análisis **estadístico** de datos. Y desde el punto de vista de la Estadística sí que podemos afirmar que R es la mejor opción y Bioconductor es su opción natural.[14] En este texto utilizamos R/Bioconductor. Es muy recomendable consultar la dirección http://www.bioconductor.org/help/workflows/ en donde podemos encontrar análisis completos de datos.

¿Qué necesitamos instalar para poder utilizar estas notas? En http://www.uv.es/ayala/docencia/tami/AllTami.R tenemos el modo de instalar todos los paquetes que se utilizan en estas notas.[15]

Funciones y paquetes. A lo largo de las notas utilizamos muchas funciones que corresponden a paquetes distintos. Una función no puede ser utilizada si no hemos cargado previamente el paquete correspondiente. Si intentamos utilizar la función sin cargar previamente el

[9] Los datos sin ningún tipo de preprocesamiento. Por ejemplo, si tenemos datos de expresión con Affymetric GeneChip se debiera de disponer de los ficheros .CEL.

[10] http://en.wikipedia.org/wiki/Literate_programming

[11] http://en.wikipedia.org/wiki/Reproducibility

[12] http://cran.r-project.org/web/views/ReproducibleResearch.html

[13] Solamente tener en cuenta que los caminos hay que modificarlos adaptándolos a donde tengamos los datos en nuestro ordenador.

[14] Sin embargo, en http://neurolex.org/wiki/Category:Resource:Gene_Ontology_Tools (en concreto buscamos dentro de la página la expresión "Statistical analysis") tenemos una buena muestra de qué podemos hacer y sobre todo con qué hacerlo.

[15] Quizás no es necesario instalarlo de una vez. Lo más conveniente es instalar los paquetes en la medida en que los necesitemos en los distintos capítulos.

paquete entonces R nos indica que no la encuentra. Con frecuencia indicaremos el nombre de la función y su paquete conjuntamente. Por ejemplo, si en el texto vemos annotate::annotation estamos refiriéndonos a la función `annotation` que se encuentra en el paquete [51, annotate].

Sobre la estructura del documento. Empezamos en I indicándo qué se entiende por Estadística de datos ómicos y con una breve introducción a R. Acabamos esta parte hablando de anotación. Necesitamos tener bases de datos (locales u online) que nos permitan saber qué a qué se refieren nuestras características (variables). ¿Qué gen o que transcrito o qué exón tengo? ¿Qué relaciones hay entre ellos? Por ejemplo: ¿qué exones componen un exon? O bien: ¿qué isoformas tengo de un gen?

En II mostramos los datos con los que trabajamos posteriormente. Se comenta el tipo de dato, dónde se puede conseguir en bases de datos públicas y cómo preprocesarlo (normalizarlo). Finalmente se ve con cierto detalle los bancos de datos que utilizamos en el resto del libro. Los datos procesados y preparados para hacer análisis estadístico los tenemos en el paquete [10, tamidata].[16] En el momento actual consideramos datos de expresión obtenidos con microarrays y con la técnica RNASeq.[17]

[16] Es un paquete propio que lo alojo en una página de la Universidad de Valencia.

[17] Aunque se pretende ampliar a otros datos ómicos.

Expresión diferencial, III, es el tema fundamental desde el punto de vista estadístico que es la opción de este manual. Se estudian los problemas de comparaciones múltiples así como técnicas de expresión diferencia en microarrays y RNASeq.

En IV hablamos de componentes principales y clustering. Desde mi (muy) personal punto de vista no son técnicas multivariantes que deban de usar en exceso en este contexto. Sin embargo, se usan y por ello se tratan aquí.[18]

[18] Faltaría un tema de clasificación supervisada que no acabo de ver su interés en este contexto.

En III nos ocupamos de la expresión diferencial gen a gen (o en lenguaje más estadístico, marginal).

Este manual se utiliza para distintos cursos. El primero es **Tecnologías de análisis molecular integrado**. [19] Se compone de los siguientes apartados indicando su contenido.

[19] El material se desarrolló en origen para esta asignatura del grado de Biotecnología de la Universidad de Valencia. Se imparte en 25 horas.

1. § 1 presenta el problema del análisis de datos ómicos.
2. En § 2 se considera la instalación de R así como la instalación de los paquetes que utilizamos en este manual. Se han de consultar las secciones de § 2.1 hasta § 2.5.
3. En § 4 se trata de datos ómicos obtenidos utilizando arrays. Nos centramos en el caso de arrays de Affymetrix. Hay que leer de § 4.1 a § 4.5 aunque el código no es necesario utilizarlo.
4. En § 5 se comentan los datos que utilizamos en el resto del manual y muchos de ellos están incluidos en [10, tamidata]. También se muestra cómo trabajar con un ExpressionSet. Las secciones que utilizamos son § 5.2, § 5.6 y § 5.7.
5. De § 7 se utilizan las secciones § 7.1, § 7.2, § 7.4, § 7.6 y § 7.7.
6. De § 8 utilizamos § 8.1, § 8.3, § 8.4. De § 8.5.1 solamente el método de Bonferroni. De § 8.6 solamente el método de Benjamini-Hochberg. Secciones § 8.7 y § 8.8.

7. De § 15 utilizamos § 15.1 y § 15.2.

A todo el material anterior hay que añadir las viñetas incluidas en el paquete [9]. Estas viñetas cubren el material de uso de R/Bioconductor.

El segundo curso que se imparte con estas notas es **Bioinformática estadística**[20] y utilizamos todo el material.

[20] Se imparte en 45 horas dentro del máster de Bioinformática de la Universidad de Valencia.

Parte I

Introducción

Capítulo 1

Estadística y datos ómicos

Estas notas se ocupan de revisar y aplicar técnicas estadísticas a datos ómicos. En lo fundamental nos ocuparemos de datos de transcripción y proteomas. No estamos interesados en el detalle exhaustivo del tratamiento de cada tipo de dato. Nos centraremos en lo que tienen en común y comentaremos lo que diferencia su tratamiento.

El énfasis de este texto es sobre **métodos estadísticos**. Evitaremos en lo posible métodos que no utilicen modelos probabilísticos. Lo que se ha dado en llamar métodos de **machine learning** o **Big Data**.[21] En este sentido es interesante recordar una frase (que subscribo) de un gran estadístico inglés Brian D. Ripley.

[21] He puesto a propósito los términos en inglés. Queda moderno. El autor no es un gran seguidor de esas técnicas nuevas. Soy un viejo desfasado que solo cree en la Probabilidad. Sin modelo no hay mucho. No hay nada.

```
fortunes::fortune(50)

##
## To paraphrase provocatively, 'machine learning is
## statistics minus any checking of models and
## assumptions'.
##    -- Brian D. Ripley (about the difference
##       between machine learning and statistics)
##       useR! 2004, Vienna (May 2004)
```

1.1 Estructura de los datos

En lo que sigue tendremos distintos tipos de datos pero con una estructura similar. En concreto, observaremos un gran número de características[22] sobre un pequeño número de muestras. Cada característica medirá la abundancia de moléculas utilizando distintos procedimientos. Esta abundancia puede estar asociada a una sonda o a un grupo de sondas en un microarray. O bien la información corresponde a un gen o a un exon, a un péptido, a una proteína. Además esta abundancia se cuantifica con distintos procedimientos (con frecuencia dependientes del fabricante del dispositivo). Hablaremos de características sin más. El número lo denotamos por N donde este valor es grande (miles). Estas características las observaremos en unas pocas muestras. El número de muestras es n (decenas con suerte). Lo básico[23] es que N es mucho mayor que n: $n \ll N$. En un contexto estadístico clásico N es

[22] Features.

[23] Y a lo largo de las notas lo repetiremos muchas veces.

el número de variables y n es el número de muestras. Y justo lo conveniente es la situación contraria. De hecho, muchos procedimientos estadísticos suponen que el tamaño de la muestra supera el número de variables. En Estadística de alto rendimiento esto no es así. Y eso da novedad a los procedimientos. Obviamente limita las posibilidades pero abre un nuevo campo de trabajo.

Las abundancias las recogemos en una matriz[24] que denotaremos por

$$\boldsymbol{x} = [x_{ij}]_{i,j=1,\ldots,n}$$

donde el valor x_{ij} nos cuantifica la abundancia de la característica i en la muestra j. [25] El valor x_{ij} será un valor (usualmente) positivo. Si corresponde un DNA microarray entonces mide un nivel de fluorescencia y tomará valores positivos.[26] Sea positivo a no un valor mayor indicará una mayor expresión del gen. Si trabajamos con datos obtenidos con RNASeq entonces tendremos conteos, esto es, número de lecturas cortas alineadas sobre un gen o sobre un exon o sobre una zona genómica de interés. En definitiva el dato primario es un número entero. Más lecturas indicará más expresión otra vez. Los valores observados en una misma fila (una misma características sobre todas las muestras) se suele decir que son un *perfil*[27] (de un modo genérico perfil de expresión).

En la matriz de expresión los valores observados para las distintas muestras son *i*ndependientes aunque posiblemente observados bajo distintas circunstancias. No son pues réplicas de una misma condición experimental pero sí se observan independientemente. Los valores de expresión para las distintas filas ya no son independientes. Por ejemplo, los genes actúan de un modo coordinado.

Los datos de la matriz de expresión no son directamente comparables. El nivel de ruido es grande y por ello se han desarrollado técnicas para corregirlo. Son métodos de corrección de fondo y normalización. Veremos algunos. En sentido estricto cuando normalizamos los datos estos dejan de ser independientes. Sin embargo, esto no se suele considerar en la literatura. Los datos después de la normalización siguen considerándose independientes por columnas (muestras) y dependientes por filas.[28]

De cada muestra tendremos información. Por ejemplo, si es una muestra control o bien corresponde a una muestra tomada bajo un tratamiento. [29] A esta información o variables que nos describen a las muestras las llamaremos los *m*etadatos o *v*ariable fenotípicas[30] Usualmente tendremos varias variables fenotípicas aunque usualmente trabajaremos con una cada vez. Denotaremos por $y = (y_1, \ldots, y_n)$ los valores observados de una variable en las n muestras. El caso maś frecuente de variable fenotípica será cuando tengamos dos grupos de muestras (casos y controles). En este caso tendremos $y_i = 1$ si es un caso e $y_i = 0$ si es un control.[31] Si tenemos más de dos grupos de muestras a comparar, por ejemplo k grupos, entonces $y_i \in \{1, \ldots, k\}$ para $i = 1, \ldots, n$.

1.2 Análisis de datos

¿Y qué vamos a hacer con la matriz de expresión y con las variables fenotípicas? Como siempre lo mejor que podamos. A veces no mucho. Las técnicas estadísticas que se utilizan son aplicaciones de procedi-

[24] Que podemos llamar matriz de expresión aunque no necesariamente hablamos de expresión de un gen pero no parece malo utilizar esta nomenclatura.

[25] Una matriz de datos en Estadística suele ser justo al contrario, esto es, con los subíndices invertidos. Es decir, la matriz transpuesta de la que vamos a utilizar aquí. En ocasiones se utiliza en la forma clásica pero es más frecuente esta forma de disponer los datos.

[26] Algunos procedimientos de procesado previo de los datos conocidos como normalización pueden dar lugar a expresiones negativas.

[27] Expression profile.

[28] Lo cual no deja de ser paradójico.

[29] La palabra tratamiento se utiliza en el sentido amplio de diseño experimental: tiempo, una cepa salvaje frente a una mutada por ejemplo.

[30] Entendido en un sentido amplio: Por variable fenotípica podemos estar entendiendo el tiempo en que se ha observado la evolución de una muestra.

[31] Los valores 0 y 1 son arbitrarios. Podemos tomar cualquier otro par de valores.

mientos diseñados en muchas ocasiones para el contexto habitual en que tienes más muestra que variables. Y se usan aquí adaptándolos con mayor o menor fortuna. Yo diría con mayor o menos sentido en ocasiones.

El problema más importante es el que se conoce como **expresión diferencial**. Nos fijamos en una variable fenotipica y en una característica (gen por ejemplo). ¿Hay asociación entre el perfil de expresion y la variable fenotípica? Veremos distintos procedimientos para responder esta pregunta *p*ara cada característica. Utilizaremos la denominación de *análisis de expresión diferencial marginal*. Sin embargo, en la literatura biológica es más hablar de *análisis de expresión diferencial gen-a-gen*.

Las distintas características son dependientes entre si. Una evaluación de la posible asociación entre cada característica y la variable fenotípica es *l*imitada. Nos puede hacer perder información sobre procesos biológicos de los cuales distintas características nos están dando información de modo que cada una de ellas recoge una variación no muy grande pero que conjuntamente es notable. En definitiva, el problema será estudiar la posible asociación entre grupos de características (grupos de filas en la matriz de expresión) y la variable fenotípica de interés. Es lo que se conoce como análisis de grupos de genes.[32]

También se verán problemas de reducción de dimensión. En concreto la técnica más utilizada, análisis de componentes principales. Es una técnica instrumental con muchas posibilidades de utilización en un contexto como es este con datos de alta dimensión.

Y clasificaremos tanto las características como las muestras como una herramienta exploratoria. Lo que se conoce como análisis cluster.

En lo que sigue se aborda también problemas de cálculo del tamaño muestral que está ligado al problema de la potencia del test.

Y todo esto siempre atendiendo al tipo de dato que estemos utilizando. Sin duda, el más desarrollado son las datos de expresión de gen utilizando microarrays. Será nuestro tipo de dato de referencia pero vamos introduciendo otros tipos de datos como pueden ser RNASeq o datos de abundancia de proteínas.

[32] Este problema puede encontrarse bajo distintas denominaciones como: Gene set analysis, gene set enrichment analysis, functional enrichment testing.

Figura 1.1: Un anuncio.

1.3 Sobre herramientas online

En este texto la opción elegida es R/Bioconductor. Es obvio que requiere un mayor conocimiento de los procedimientos que se utilizan pero nos dan una enorme flexibilidad para trabajar. Podemos preprocesar la información y procesarla como queramos. Obviamente hay que saber cómo hacerlo. Y de esto va el texto. Sin embargo, cada vez más hay herramientas en línea[33] que merecen explorarse (aunque no sea nuestra opción). En lo que sigue mostramos una enumeración con algún comentario.[34] En particular se consideran:

GEPIA http://gepia.cancer-pku.cn/index.html Es una buena implementación para estudios de cáncer. Permite el análisis tanto de gen a gen (o marginal) como el análisis de grupos de genes. Todo orientado al estudio de los cánceres. Los datos no los proporciona el usuario. Realmente utilizan datos de los proyectos TCGA y GTEx.

[33] Online en La vida moderna.

[34] No tengo un gran conocimiento de su manejo porque insisto no es mi opción personal. Las manejo para ver qué ofrecen pero siempre encuentras que harías las cosas de otra manera. Cosas de la edad.

NetworkAnalyst http://www.networkanalyst.ca Es de particular interés la tabla 1 en [150, pág. 824] en donde se muestra una comparativa entre distintas herramientas online.

DAVID http://david.abcc.ncifcrf.gov.

g:Profiler http://biit.cs.ut.ee/gprofiler/.

InnateDB con Cytoscape http://www.innatedb.ca y http://www.cytoscape.org.

Gitools http://www.gitools.org.

1.4 Paquetes transversales

Algunos paquetes que vamos a manejar pretenden cubrir una gran cantidad de contenidos que vamos a tratar: lectura de la información generando las clases adecuadas (como Biobase::ExpressionSet o SummarizedExperiment::RangedSummarizedExperiment); normalización; análisis de expresión diferencial marginal o gen a gen; análisis de enriquecimiento para grupos o bien para grafos. En fin, tienen una pretensión de análisis global. Usualmente lo que hacen es utilizar distintos paquetes según realiza una u otra funcionalidad. Dentro de esta categoría entran [49, EnrichmentBrowser] o [**R-piano**]. Los usaremos aunque siempre suponen una limitación en el análisis que se realiza. Soy más partidario de utilizar funcionalidades concretas del paquete más que utilizarlo para todo el análisis.

1.5 Jerga

En un campo como es la Bioinformática un problema mayor es el lenguaje que se utiliza. Un texto de Estadística o un texto de Genética tiene una jerga consolidada. Son campos de trabajo maduros.[35] Prácticamente cada campo de investigación va desarrollando una jerga[36] en donde suele haber una parte interesada. Impedir el acceso a los que no son de este campo. Aquí el problema es grave porque se mezclan jergas al mismo tiempo que se desarrolla una nueva. Para intentar evitar esta barrera he incluido un glosario al final del texto que intenta aclarar los términos y sirva de referencia rápida. Sin embargo, quizás la mayor dificultad de esta disciplina sea este, entender la jerga de los demás.[37]

[35] Sin ser peyorativo, campos de investigación viejos.

[36] Mejor una jerigonza.

[37] En mi caso de los biólogos. A los informáticos los entiendo. Hablan menos de hecho.

1.6 Datos ordenados

Un problema que comparte el análisis de datos ómicos con cualquier otro problema de análisis de datos es tener unos datos de entrada *ordenados*[38] así como unas salidas de las distintas funciones que los analizan que sean ordenadas, fácilmente interpretables y que puedan ser una entrada ordenada para un procedimiento posterior. El término utilizado en [143] es **tidy data**. En este texto utilizaremos distintas herramientas que prestan atención a este aspecto del análisis que ahorra mucho tiempo de trabajo y muchísimos quebraderos de cabeza. Entre otros paquetes utilizaremos [12, biobroom] y [114, broom].

[38] Tidy data.

1.7 Bibliografía

A lo largo del manual se presta mucha atención a citar con precisión las referencias originales del material. La comprensión precisa de las técnicas supone la consulta de la referencia original.[39] Un libro que trata cómo hacer las cosas con R/Bioconductor pero no lo que se hace o porqué se hace es [123]. Es una guía de uso muy bien elaborada. Un texto con un objetivo similar al nuestro es [62]. Un manual online que sigue una línea muy similar a este es http://genomicsclass.github.io/book/.

[39] Hay una peligrosa tendencia a creer que leyendo un resumen se conoce la técnica. No es cierto. Las referencias bibliográficas siempre hay que consultarlas.

Capítulo 2

R y Bioconductor

Nuestra opción de trabajo es la utilización de R y Bioconductor. En adelante, hablaremos de R/Bioconductor. No es la única opción. Dos buenas opciones de software libre son Babelomics o DAVID Bioinformatics. Y una opción comercial es Partek Genomics Suite. Sin embargo, cuando hablamos de análisis estadístico de datos de alto rendimiento sí que, en el momento actual, es en mi opinión la mejor opción. Y en este manual sobre todo hablamos de análisis de datos.

2.1 Sobre R y su instalación

2.1.1 De cómo instalarlo

Instalación Los pasos a seguir son los siguientes:

1. Bajamos el programa de la siguiente dirección http://cran.r-project.org/.
2. Elegimos versión.[40]
3. Una vez hemos bajado el paquete se instala ejecutándolo con las opciones por defecto.

[40] Se recomienda cualquier versión de Linux. Si utilizas Windows (en esto el mundo es bastante libre) elige la correspondiente versión para Windows.

Inicio de una sesión En el escritorio tenemos el icono de R. Simplemente clicando[1] el icono iniciamos la sesión de trabajo.

Instalación de un paquete R tiene muchos paquetes que extienden el R base que acabamos de instalar. De hecho, es casi imposible realizar un análisis estadístico por sencillo que sea sin utilizar paquetes adicionales de R. Vamos a instalar el paquete [138, UsingR]. Es un paquete con herramientas para la enseñanza de Estadística básica. La opción más simple es escribir en línea de comandos.

```
install.packages(``UsingR'')
```

Cargando un paquete Una vez instalado el paquete, para poder usar las funciones o datos que contenga, debemos *cargarlo* mediante

[1] Dios nos perdone por usar esta palabra aunque no creo que lo haga.

```
library(UsingR)
```

Ahora podemos utilizar las extensiones que proporciona a R este paquete.

A lo largo de estas notas iremos aprendiendo a utilizar (lo que necesitamos de) R según lo necesitemos. Sin embargo, una referencia rápida a la sintaxis de R la podemos encontrar en http://cran.r-project.org/doc/contrib/Short-refcard.pdf.[2] .

Otra opción para instalar es utilizar [113, pacman]. La instalación de [138, UsingR] sería[41]

[41] Elegimos como repositorio https://ftp.cixug.es/CRAN.

```
pacman::p_install("UsingR",repos="https://ftp.cixug.es/CRAN/")
```

2.2 Lo primero con R

De cómo trabajar con R

Hay dos formas de trabajar con R. La primera opción es utilizar un editor en el que escribimos código de R. Lo copiamos y luego lo pegamos en la línea de comandos de R.[3] Dentro de esta manera de trabajar podemos utilizar distintas editores (que llevan herramientas para facilitarnos el trabajo).

Con el editor de R El propio programa lleva un editor incorporado. Es básico pero suficiente. Es la opción que utilizaremos en las clases prácticas.

RStudio Quizás ahora mismo sea la mejor opción. Es un programa que incorpora el editor, nos muestra las salidas, los gráficos y la historia previa. De manejo muy simple. http://rstudio.org/.

Figura 2.1: Un anuncio.

De cómo conseguir ayuda con R

Supongamos que buscamos ayuda sobre las opciones de la función que nos dibuja un histograma, `hist`. Lo más simple es utilizar la ayuda en html. Utilizamos la siguiente función.

```
help.start()
```

Vemos que nos abre el navegador y nos ofrece distintas opciones. Quizás la opción más simple sea utilizar la herramienta de búsqueda. Otra opción es

```
?hist
```

O simplemente,

[2] Una buena idea es imprimirla y tenerla a mano.

[3] Es la opción más educativa en donde aprendemos realmente a trabajar con el programa.

```
help(hist)
```

2.3 Una sola muestra

Supongamos que consideramos una técnica ómica (microarray, RNAseq). Estamos estudiando la expresión de 10 genes y los valores observados son los siguientes.

```
##  [1] 3 6 3 6 7 1 4 7 4 4
```

Podemos leer estos datos con

```
x = c(3,6,3,6,7,1,4,7,4,4)
```

Tenemos ahora un objeto llamado `x`

```
class(x)

## [1] "numeric"
```

que es un vector formado por números. ¿Cómo puede ver su contenido?

```
x

##  [1] 3 6 3 6 7 1 4 7 4 4
```

¿Cuál es el primer valor de este vector de datos?

```
x[1]

## [1] 3
```

¿Y el que ocupa la posición 7?

```
x[7]

## [1] 4
```

Podemos ver los datos que están entre el 3 y el 7. Para ello fijémonos en el siguiente código.

```
3:7

## [1] 3 4 5 6 7
```

Cuando ponemos dos enteros separados por `:` nos devuelve todos los enteros entre el primero y el segundo. Lo mismo lo podemos hacer con

```
seq(3,7,1)

## [1] 3 4 5 6 7
```

Ahora podemos ver los datos que ocupan estas posiciones en el vector `x`.

```
x[3:7]

## [1] 3 6 7 1 4
```

Podemos tener interés en saber los valores de los datos que ocupan las posiciones 1, 3 y de la 5 a la 8. Estas posiciones las podemos obtener con

```
c(1,3,5:8)

## [1] 1 3 5 6 7 8
```

y los valores de `x` serían

```
x[c(1,3,5:8)]

## [1] 3 3 7 1 4 7
```

Puede que nuestro interés en ver los datos no venga dado por la posición que ocupan sino por su valor. Por ejemplo, queremos saber cuántos de estos datos superan o son iguales a 4. ¿Cómo lo hacemos? Lo lógico es comparar los valores de `x` con 4. Lo hacemos con

```
x >= 4

##  [1] FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE
##  [8]  TRUE  TRUE  TRUE
```

Vemos que nos devuelve un vector diciéndonos si es cierta o no la condición que hemos preguntado, si es mayor o igual a 4. Pero: ¿qué valores son? Si hacemos

```
x[x >= 4]

## [1] 6 6 7 4 7 4 4
```

Nos devuelve los datos que ocupan las posiciones donde se daba la condición, donde la condición era cierta. Podemos saber qué valores toman los datos que son mayores que 37 con

```
x[x > 6]

## [1] 7 7
```

o bien los datos que son mayores que 4 y menores o iguales que 6.

```
x[x > 4 & x <= 6]

## [1] 6 6
```

Podemos querer que los casos que estamos seleccionando estén en un nuevo vector.

```
y = x[x > 4 & x <= 6]
```

Los valores de y son

```
y

## [1] 6 6
```

Podemos querer los que son menores o iguales a 4 **o** mayores que 6.

```
x[x <= 4 | x > 6]

## [1] 3 3 7 1 4 7 4 4
```

¿Y los que son iguales a 4? Observar el uso de ==.

```
x[x == 4]

## [1] 4 4 4
```

¿Y en qué posiciones del vector tenemos los valores iguales a 4.

```
which(x == 4)

## [1]  7  9 10
```

[42]

[42] El resto de operadores lógicos los encontramos en la ayuda de *Logical Operators*: & (&&) corresponde con la intersección, | (||) corresponde con la unión y la negación de una condición la obtenemos con !.

De cómo ordenar un vector Supongamos que queremos ordenar el vector x.

```
sort(x)

##  [1] 1 3 3 4 4 4 6 6 7 7
```

Nos devuelve los valores ordenados. Sin embargo, con frecuencia necesitamos saber la posición que ocupaban en el vector original estos valores.

```
sort(x,index.return = TRUE)

## $x
##  [1] 1 3 3 4 4 4 6 6 7 7
##
## $ix
##  [1]  6  1  3  7  9 10  2  4  5  8
```

Hemos ordenado el vector x de menor a mayor. Vamos a ordenar de mayor a menor.

```
sort(x,decreasing = TRUE,index.return = TRUE)

## $x
##  [1] 7 7 6 6 4 4 4 3 3 1
##
## $ix
##  [1]  5  8  2  4  7  9 10  1  3  6
```

* **Ej. 1** — Introducir los siguientes datos en R.

```
##   [1]  9  5  6  5  2  8  3  2  4  9  8  6  6 11  6
##  [16]  6  5  5  4  3  9  8  6  7  1  5  6  3  4  3
##  [31]  3  4  4  4  3  3  3  5  4  7  2  5  7  2  5
##  [46]  3  3  6  8  4  6  2  4  4  7  5  7  7  7  4
##  [61]  6  6  6  0  5  3  4  5  4  2  3  6  4  7  2
##  [76]  4 10  8  8  3  3  6  4  6  4  3  7  2  5  5
##  [91]  5  4  5  9  5  8  8  5  4  3
```

Se pide:

1. ¿Cuál es el valor medio, la varianza y la desviación estándar de estos valores?
2. ¿Cuántos de los valores son 3?
3. ¿Cuál es el rango intercuartílico?
4. Ordena el vector y calcula las sumas acumuladas de los valores ordenados con la función base::cumsum.

* **Ej. 2** — Ejecutad el siguiente código.[43]

```
x = rnorm(23489,mean=45,sd=2.3)
```

[43] Estamos generando valores con distribución normal con media 45 y desviación estándar 2.3. Simplemente son unos datos para trabajar con ellos.

Se pide:

1. ¿Cuántos valores de `x` son mayores o iguales de 39.4?
2. ¿Cuántos valores de `x` son estrictamente menores que 46?
3. ¿Cuántos valores de `x` son estrictamente menores que 46 y mayores o iguales de 39.4?
4. ¿Cuántos valores son tales que su parte entera (por defecto) es igual a 40? Se puede utilizar también la función base::floor.

2.3.1 Varias muestras

Obviamente no tendremos habitualmente una sola muestra sino que tendremos más de una muestra. En cada una de ellas tendremos observada la expresión de los mismos genes. Supongamos que tenemos la expresión 10 genes en 4 muestras. Simulamos estos datos.

```
set.seed(123)
(x = matrix(rpois(10*4,lambda=5),nrow=10))
```

```
##       [,1] [,2] [,3] [,4]
##  [1,]    4    9    8    9
##  [2,]    7    5    6    8
##  [3,]    4    6    6    6
##  [4,]    8    5   11    7
##  [5,]    9    2    6    1
##  [6,]    2    8    6    5
##  [7,]    5    3    5    6
##  [8,]    8    2    5    3
##  [9,]    5    4    4    4
## [10,]    5    9    3    3
```

¿Qué tipo de dato es `x`?

```
class(x)

## [1] "matrix" "array"
```

Es una matriz. Podemos darle nombre a sus filas y columnas.

```
colnames(x) = c("sample1","sample2","sample3","sample4")
```

[44] El siguiente código es equivalente al anterior y más simple.

[44] La función base::c simplemente construye un vector y en cada posición tiene el nombre de la muestra. Podemos concatenar también números.

```
colnames(x) = paste0("sample",1:4)
```

Además sabemos que en cada fila tenemos la expresión correspondiente a un gen distinto. De momento, supondremos que son "gene1", "gene2", etc. Vamos a cambiarle el nombre las filas utilizando el código que acabamos de ver.

```
rownames(x) = paste0("gene",1:10)
```

¿Qué tenemos ahora?

```
x

##        sample1 sample2 sample3 sample4
## gene1        4       9       8       9
## gene2        7       5       6       8
## gene3        4       6       6       6
## gene4        8       5      11       7
## gene5        9       2       6       1
## gene6        2       8       6       5
## gene7        5       3       5       6
## gene8        8       2       5       3
## gene9        5       4       4       4
## gene10       5       9       3       3
```

Podemos ver la expresión del gen en la fila 6 y en la columna 2 con

```
x[6,2]

## [1] 8
```

o bien con

```
x["gene6","sample2"]

## [1] 8
```

Si queremos ver la expresión de los genes en la muestra 2 podemos hacerlo con

```
x[,2]

##  gene1  gene2  gene3  gene4  gene5  gene6  gene7
##      9      5      6      5      2      8      3
##  gene8  gene9 gene10
##      2      4      9
```

o

```
x[,"sample2"]

##  gene1  gene2  gene3  gene4  gene5  gene6  gene7
##      9      5      6      5      2      8      3
##  gene8  gene9 gene10
##      2      4      9
```

Ej. 3 — Ejecutad el siguiente código. Nos genera una matriz donde los valores siguen una distribución normal con media 34 y desviación estándar 3.21.

```
x = matrix(rnorm(2345*122,mean=34,sd=3.21),nrow=2345)
```

Se pide:

1. ¿Cuántas columnas tiene la matriz?
2. ¿Qué columna tiene la media más alta? Utilizad las funciones base::apply y base::which.max.
3. ¿Qué fila tiene la media más pequeña? Utilizad las funciones base::apply y base::which.min.
4. ¿Qué fila tiene el menor rango intercuartílico? Utilizad las funciones base::apply y base::which.min.
5. ¿En cuántas ocasiones la primera columna es mayor o igual que la segunda columna de la matriz?
6. Determinamos la media de cada una de las filas. ¿Cuántas filas son tales que la columna 45 es mayor o igual que el vector de medias que acabamos de calcular?

2.4 Datos golub

Los datos golub son banco de datos de expresión de genes que vamos a utilizar. Aparecen en multtest::golub y fueron utilizados en [60]. Los datos son los niveles de expresión de 3051 genes (que aparecen en filas) para 38 pacientes de leucemia. De estos pacientes, 27 de ellos tienen leucemia linfoblástica aguda (ALL) y los restantes 11 tienen leucemia mieloide aguda (AML). El tipo de tumor viene indicado por el vector *golub.cl* donde ALL corresponde con 0 y AML corresponde con 1. Los nombres de los genes los tenemos en *golub.gnames*. Cargamos los datos.

```
data(golub,package = "multtest")
```

Al cargar estos datos hemos leído varias cosas:

golub.cl Es un vector numérico que toma valores 0 y 1 indicando si la muestra ha sido obtenida de un enfermo con leucemia linfoblástica aguda (0) o leucemia mioloide aguda (1).

golub Es una matriz de expresión donde cada fila corresponde con un gen y cada columna con una muestra distinta.

golub.gnames Es una matriz con el mismo número de filas de la matriz golub de modo que la fila i-ésima de golub.gnames nos da información sobre el gen de cuyos niveles de expresión aparecen en la fila i-ésima de la matriz golub.

Empezamos jugando con el vector `golub.cl`. Podemos ver los valores.

```
golub.cl

##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [24] 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

¿Qué tipo de dato tenemos?

```
class(golub.cl)

## [1] "numeric"
```

Vemos que son números. Son números que, realmente, indican la pertenencia a un tipo de cáncer u otro. Por ello es más natural definirlo como un vector tipo `factor`.[45] Lo hacemos con base::factor.

[45] Como en R codificamos las variables categóricas.

```
(golub.fac = factor(golub.cl,levels=0:1,labels=c("ALL","AML")))

##  [1] ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL
## [19] ALL ALL ALL ALL ALL ALL ALL ALL ALL AML AML AML AML AML AML AML AML AML
## [37] AML AML AML AML AML
## Levels: ALL AML
```

¿Cuántas muestras tenemos de cada tipo? Un par de opciones para hacerlo son utilizando la función base::table.

```
table(golub.fac)

## golub.fac
## ALL AML
##  27  11
```

o utilizando la función base::summary.

```
summary(golub.fac)

## ALL AML
##  27  11
```

Si queremos mostrar la distribución de los dos tipos en forma de diagrama de barras lo podemos hacer con (ver figura 2.2)

```
pacman::p_load(ggplot2)
df0 = data.frame(golub.fac)
png(paste0(dirTamiFigures,"RBio39.png"))
ggplot(df0,aes(x=golub.fac))+geom_bar()
dev.off()
```

Sustituye en el código anterior `golub.fac` por `golub.cl` y verás que el resultado no es el deseable. ¿Por qué? Quizás los números no es una buena forma de representar categorías o clases.

Nos fijamos en `golub`. ¿Qué tipo de dato es?

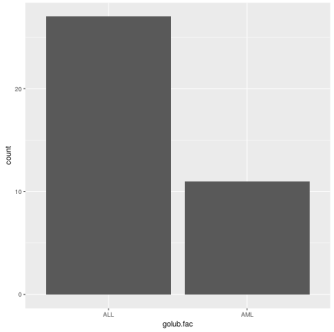


Figura 2.2: Diagrama de barras con las frecuencias absolutas de cada tipo de leucemia.

```
class(golub)

## [1] "matrix" "array"
```

Es una matriz. En una matriz lo primero es saber cuántas filas (características aquí) tenemos

```
nrow(golub)

## [1] 3051
```

El número de columnas corresponde con el número de muestras.

```
ncol(golub)

## [1] 38
```

También podemos obtener las dimensiones de la matriz conjuntamente con

```
dim(golub)

## [1] 3051   38
```

En la fila i y columna j tiene una cuantificación de la expresión del gen i-ésimo[46] en la j-ésima muestra de nuestra experimentación. Por ejemplo, el gen en fila 2000 y columna 12 tiene expresión

[46] Ya veremos más adelante de qué gen estamos hablando.

```
golub[2000,12]

## [1] 0.19595
```

[47] Y todos los niveles de este gen lo tendremos con

[47] Estos datos han sido preprocesados partiendo de los datos originales. Pueden aparecer valores negativos. En § 5.3 se explica y se entiende el porqué del valor negativo. Lo importante valor menor indica menor expresión, valor mayor indica mayor expresión.

```
golub[2000,]

##  [1]  0.73124 -0.19598  0.51981 -0.54371  0.55596
##  [6]  1.40683  0.79772  0.59493  0.99503  0.39529
## [11]  0.09834  0.19595  0.85017 -1.39979  1.09789
## [16] -0.74362  0.44207  0.27698 -0.04128 -1.60767
## [21] -1.06221 -1.12665  0.47863 -0.44014  0.22286
## [26]  0.42795  0.65427  0.07257 -0.28093 -0.20985
## [31]  0.05160 -1.44434 -0.17118 -1.34158  0.92325
## [36] -0.21462 -1.34579  1.17048
```

[48] Expression profile.

Estos valores reciben el nombre de *perfil de expresión*.[48] Podemos representar el perfil de expresión (figura 2.3). Definimos qué van en el eje de abscisas y en el eje de ordenadas.

```
pacman::p_load(ggplot2)
muestra = 1:ncol(golub) ## En abscisas el número de la muestra
y2000 = golub[2000,] ## En ordenadas su expresión
df = data.frame(muestra = 1:ncol(golub),y2000= golub[2000,])
png(paste0(dirTamiFigures,"RBio47.png"))
ggplot(df,aes(x = muestra,y= y2000))+geom_point()
dev.off()
```

Lo podemos ver en figura 2.3(a).

Si queremos ver las expresiones correspondientes a los pacientes ALL lo podemos hacer con [49]

[49] Observermos que "==" se utiliza para comparar e indica TRUE O FALSE según sea cierta o falsa la condición.

```
golub[2000,golub.fac == "ALL"]

##  [1]  0.73124 -0.19598  0.51981 -0.54371  0.55596
##  [6]  1.40683  0.79772  0.59493  0.99503  0.39529
## [11]  0.09834  0.19595  0.85017 -1.39979  1.09789
## [16] -0.74362  0.44207  0.27698 -0.04128 -1.60767
## [21] -1.06221 -1.12665  0.47863 -0.44014  0.22286
## [26]  0.42795  0.65427
```

y los correspondientes a los pacientes AML.

```
golub[2000,golub.fac == "AML"]

##  [1]  0.07257 -0.28093 -0.20985  0.05160 -1.44434
##  [6] -0.17118 -1.34158  0.92325 -0.21462 -1.34579
## [11]  1.17048
```

Supongamos que volvemos a representar el perfil de expresión del gen en la fila 2000, `y2000`, pero pretendemos diferenciar con un color distinto la condición que nos indica el tipo de leucemia, golub.fac. Lo podemos hacer con (figura 2.3(b)):

```
df = data.frame(muestra = 1:ncol(golub),y2000= golub[2000,],
                tipo = golub.fac)
png(paste0(dirTamiFigures,"RBio53.png"))
ggplot(df,aes(x = muestra,y= y2000,color=tipo))+geom_point()
dev.off()
```

Otra opción sería representar separadamente[50] los datos correspondientes a cada tipo de leucemia (figura 2.3(c)). También podemos hacerlo con

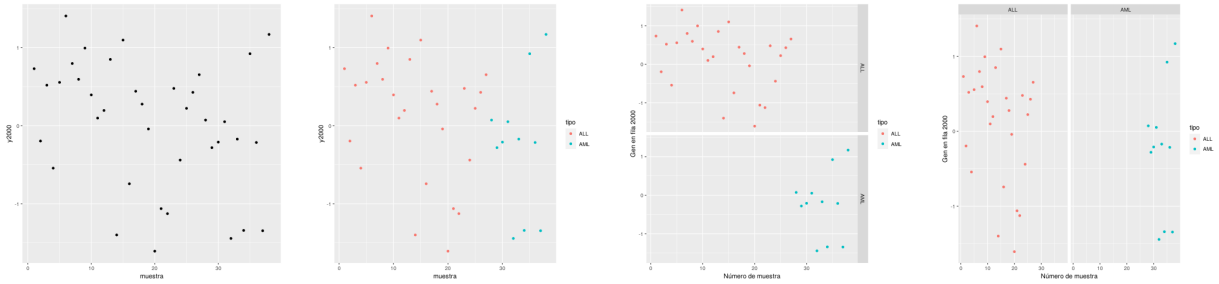
[50] En facets distintas.

```
df = data.frame(muestra = 1:ncol(golub),y2000= golub[2000,],
                tipo = golub.fac)
png(paste0(dirTamiFigures,"RBio56.png"))
ggplot(df,aes(x = muestra,y= y2000,colour=tipo))+geom_point() +
    xlab("Número de muestra") + ylab("Gen en fila 2000") +
    facet_grid(tipo~.)
dev.off()
```

Podemos colocar los distintos subdibujos (o facets) horizontalmente (2.3(d)).

```
df = data.frame(muestra = 1:ncol(golub),y2000= golub[2000,],
                tipo = golub.fac)
png(paste0(dirTamiFigures,"RBio57.png"))
ggplot(df,aes(x = muestra,y= y2000,colour=tipo))+geom_point() +
    xlab("Número de muestra") + ylab("Gen en fila 2000") +
    facet_grid(. ~ tipo) #Modificamos esta línea
dev.off()
```

En gráficos (en todo) la simplicidad es un valor. De las tres figuras anteriores quizás la figura 2.3(b) sea la más adecuada.



(a) (b) (c) (d)

Figura 2.3: (a) Perfil de expresión del gen en la fila 2000. (b) Perfil de expresión del gen en fila 2000 diferenciando el tipo de leucemia. (c) Perfil de expresión del gen en fila 2000 diferenciando el tipo de leucemia mediante un color distinto. Utilizamos distintos dibujos (facets) alineados en vertical. (d) Perfil de expresión del gen en fila 2000 diferenciando el tipo de leucemia mediante un color distinto. Utilizamos distintos dibujos (facets) alineados en vertical.

¿De qué genes estamos hablando? ¿A qué gen corresponde esta fila? La información la podemos obtener con

```
golub.gnames[2000,]
```

¿Qué tipo de datos tenemos ahora?

```
class(golub.gnames)

## [1] "matrix" "array"
```

Es una matriz con dimensiones

```
dim(golub.gnames)

## [1] 3051    3
```

Cada fila nos da información con el gen del cual tenemos su perfil de expresión en la matriz golub. En concreto, ¿qué tenemos en la fila 2000?

```
golub.gnames[2000,]

## [1] "4544"          "CDC21 HOMOLOG"
## [3] "X74794_at"
```

Los datos originales de la experimentación tenían más genes estudiados. Se hizo una selección (§ 5.3). Este gen ocupaba la fila 4544 en la matriz de expresión con todos los genes. Su nombre es

```
golub.gnames[2000,2]

## [1] "CDC21 HOMOLOG"
```

El identificador de la sonda Affymetrix o AffyID (§ 4.2) ocupa la tercera columna y vale, para este gen,

```
golub.gnames[2000,3]

## [1] "X74794_at"
```

Vamos a modificar la matriz `golub` de modo que recoja la información que tenemos de las filas (qué genes son) y de las columnas (qué tipo de leucemia). Para ello podemos aplicar las funciones `rownames` y `colnames` a dichas matrices.

1. Identificamos las filas con el identificador Affy.

```
rownames(golub) = golub.gnames[,3]
```

2. Describimos las columnas con el tipo de leucemia.

```
colnames(golub) = golub.fac
```

Tenemos la información de un modo más compacto.

2.5 La función apply

Vamos a explorar los datos golub utilizando la función base::apply.[51]

¿Qué pretendemos hacer? Vamos a realizar dos dibujos. En abscisas pretendemos dar una medida de localización (media o mediana) de la expresión del gen en el primer tipo de leucemia, ALL. En ordenadas lo mismo pero con el segundo tipo, AML.

Primero hemos de calcular estos valores. La función que calcula la media (muestral) de un vector de números es base::mean. Por ejemplo, la media del perfil de expresión del gen en la fila 2000 sería

```
mean(golub[2000,])

## [1] 0.02080211
```

También parece una buena opción cuantificar el valor alrededor de donde se observan las expresiones mediante la mediana. La tenemos con

```
median(golub[2000,])

## [1] 0.147145
```

Vemos que son muy distintos indicando asimetría.[52]

¿Y cómo lo hacemos para todos los genes de una vez? La función base::apply lo hace. Primero seleccionamos las partes de la matriz `golub` con las muestras `ALL` y con las muestras `AML`.

```
golub.ALL = golub[,golub.fac == "ALL"]
golub.AML = golub[,golub.fac == "AML"]
```

Ahora utilizamos base::apply. El primer argumento es la matriz con la que trabajamos, el segundo argumento indica si la función a

[51] Otras muy relacionadas que utilizaremos son base::lapply y base::sapply.

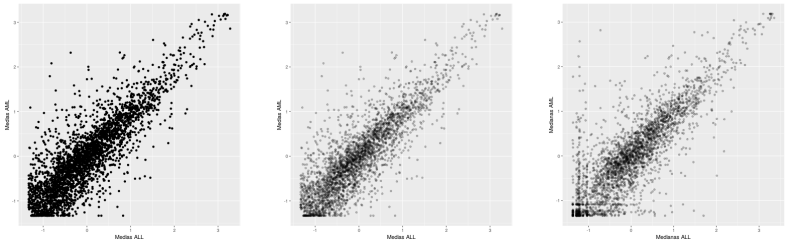
[52] Cuando para un conjunto de datos no hay coincidencia entre media y mediana lo que sucede es que no hay una disposición simétrica de los datos alrededor de estas medidas de localización.

aplicar es por filas (1) o por columnas (2). El tercer argumento es la función que queremos aplicar. Calculamos la media y la mediana para cada submatriz de golub.

```
ALL.mean = apply(golub[,golub.fac == "ALL"],1,mean)
AML.mean = apply(golub[,golub.fac == "AML"],1,mean)
ALL.median = apply(golub[,golub.fac == "ALL"],1,median)
AML.median = apply(golub[,golub.fac == "AML"],1,median)
```

Representamos las medias de ALL (abscisas) frente a las medias AML (ordenadas) (figura 2.4(a)).

```
df = data.frame(x0 = ALL.mean, y0 = AML.mean)
png(paste(dirTamiFigures,"RBio72.png",sep=""))
p = ggplot(df,aes(x=x0,y=y0))+geom_point()
p + xlab("Medias ALL") + ylab("Medias AML")
dev.off()
```



(a) (b) (c)

Figura 2.4: Medias en muestras AML frente a medias en muestras ALL para datos multtest::golub. En (a) tenemos los puntos tal cual y en (b) utilizamos un sombreado. (c) Medianas en muestras AML frente a medianas en muestras ALL para datos multtest::golub. sustituimos la media por la mediana.

En la figura 2.4(a) hay muchos puntos representados. Es un scatterplot o diagrama de puntos muy denso. Esto será habitual en lo que sigue pues trabajamos con grandes bancos de datos. Tantos puntos que se van superponiendo nos impide ver en qué zonas hay una mayor densidad de puntos. No es útil un dibujo así. Quizás ayude un sombreado que esté relacionado con la densidad local de puntos que representamos. Lo conseguimos con el argumento `alpha` (figura 2.4(b)).[53]

[53] Este argumento alpha indicado como una fracción 1/4 en el ejemplo indica que cuando se superpongan 4 puntos entonces se representa el punto completamente negro. Por debajo se utilizan grises.

```
png(paste0(dirTamiFigures,"RBio73.png"))
ggplot(df,aes(x=x0,y=y0))+geom_point(alpha=.25)+xlab("Medias ALL") +
    ylab("Medias AML")
dev.off()
```

Reproducimos el dibujo sustituyendo la media por la mediana (figura 2.4(c)).

```
df = data.frame(x0 = ALL.median, y0 = AML.median)
png(paste(dirTamiFigures,"RBio75.png",sep=""))
p = ggplot(df,aes(x=x0,y=y0))+geom_point(alpha=.25)
p + xlab("Medianas ALL") + ylab("Medianas AML")
dev.off()
```

* **Ej. 4** — Utilizando la matriz de expresión de los datos `golub` se pide:

1. Calcular para cada gen la expresión media sin considerar el tipo de cáncer.
2. Calcular para cada gen la desviación estándar.
3. Representar gráficamente un dibujo que, para cada gen, nos muestre en abscisas la expresión media y en ordenadas la desviación estándar.
4. Utilizando la función grDevices::png guardar el dibujo anterior en un fichero externo.

* **Ej. 5** — Con los datos golub se pide:

1. Calcular la expresión media para cada una de las condiciones definidas por la variable golub.cl.
2. Representar una media frente a la otra.
3. Con la función *abline()* añadir la recta $y = x$ así como las rectas $y - x = \delta$ e $y - x = -\delta$ donde δ es un valor que iremos variando.
4. ¿Qué nos indicarían los genes tales que sus puntos asociados están fuera de la región $\{(x, y) : |y - x| > \delta\}$.

* **Ej. 6** — Consideremos los datos `golub`. Se pide:

1. Determinar la desviación estándar de las expresiones de cada gen.
2. Representar un histograma de estas desviaciones estándar.
3. Calcular el percentil de orden 0.9 de las desviaciones calculadas en el punto anterior. Denotemos este valor por $q_{0.9}$.
4. Utilizando la función *abline* añadir al dibujo del apartado 2 una línea vertical cuya abscisa coincida con el valor $q_{0.9}$.
5. Seleccionar aquellos genes cuya desviación estándar sea mayor que el valor $q_{0.9}$ del punto anterior.

* **Ej. 7** — El siguiente código genera una matriz y la guarda en `x`.

```
set.seed(123)
x = matrix(runif(500),nrow=100)
```

Ejecuta en código anterior. Las preguntas que siguen se refieren a la matriz `x`.

1. ¿Cuántas columnas tiene la matriz anterior?
2. Calcula los valores máximos en cada una de las columnas de `x`.
3. Calcula los valores mínimos de las filas de `x`.
4. ¿En qué fila tenemos el mínimo más pequeño de los mínimos por fila calculados en 3?
5. ¿En qué fila tenemos el mínimo más grande de los mínimos por fila calculados en 3?
6. Determina el percentil de orden 0.34 de la primera columna de la matriz `x`.
7. Calcular el percentil de orden 0.23 de las medias por fila de la matriz `x`.

8. ¿Cuál es el mínimo, máximo y valor medio de `x`?
9. Representar un histograma de los datos de la primera columna de `x` utilizando [145, ggplot2].
10. Representar un estimador kernel de la densidad de los datos de la primera columna de `x` utilizando [145, ggplot2].
11. ¿Cuántos elementos de la matriz `x` son mayores o iguales a 0.12?
12. Para cada fila de la matriz `x` calcula cuántos valores son menores o iguales a 0.12.
13. De los valores calculados en 12 determina el rango intercuartílico.
14. ¿Cuántas filas son tales que la primera columna es mayor o igual a la segunda?

** **Ej. 8** — Construir un `data.frame` utilizando los datos multtest::golub que tenga las siguientes caracteríscas.

1. Cada variable (columna) corresponda a una sonda y cada fila corresponda con una muestra distinta.
2. Los nombres de las variables han de ser el nombre "probe1", "probe2", etc.
3. Añadir una variable (que llamaremos `type` que indique el grupo de leucemia al que pertenece cada muestra. Esta variable ha de ser de tipo `factor` y las etiquetas (o `labels`) han de ser `AML` (leucemia mieloide aguda) y `ALL` (leucemia linfoblástica aguda).

[54]

[54] En lo que sigue veremos una clase llamada ExpressionSet mucho más adecuada que el data.frame para este tipo de datos.

2.6 Listas

Las listas son muy importantes en R. En este manual las usaremos, entre otras cosas, cuando trabajamos con grupos de genes. Supongamos que estamos trabajando con un grupo de 10 genes y los representamos con un número. Por tanto nuestro universo de genes es el conjunto $\{1, \ldots, 10\}$. De estos diez genes suponemos que se ha descrito asociación para los siguientes grupos:

```
geneset1 = c(1,4)
geneset2 = c(2,5,7,8)
geneset3 = c(1,5,6)
```

¿Cómo almacenamos esta información? Lo mejor es una lista. La empezamos creando.

```
(genesets = vector("list",3))

## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
```

De momento la tenemos vacía. Hacemos que el primer elemento de la lista contenga al primer conjunto.

```
genesets[[1]] = geneset1
```

También para el segundo y tercero.

```
genesets[[2]] = geneset2
genesets[[3]] = geneset3
```

Podemos ver lo que tenemos ahora.

```
genesets

## [[1]]
## [1] 1 4
##
## [[2]]
## [1] 2 5 7 8
##
## [[3]]
## [1] 1 5 6
```

Parece natural darle un nombre a cada elemento. Supongamos que pretendemos llamarlos `set1`, `set2` y `3` respectivamente.

```
names(genesets) = paste0("set",1:3)
```

Los genes tienen distintas denominaciones como vemos más adelante. De momento, nos conformamos con llamarlos `gene1`, etc. Construimos un vector de nombres.

```
genenames = paste0("gene",1:10)
```

Convertimos los elementos de la lista utilizando los nombres en lugar de las posiciones. Por ejemplo, para el primero elemento sería

```
genesets[[1]] = genenames[geneset1]
```

Y lo mismo para los otros.

```
genesets[[2]] = genenames[geneset2]
genesets[[3]] = genenames[geneset3]
```

Ya tenemos una lista con los grupos de genes descritos.

2.7 Funciones con R

No podemos trabajar con R sin utilizar funciones.[55] La mayor potencia de R es que podemos definir funciones para extenderlo.

[55] Es un lenguaje funcional.

La estructura básica de una función en R es

```
nombre.funcion = function(argumentos){
 CODIGO
 resultado
}
```

Supongamos que queremos calcular la media y desviación estándar de un vector de datos. La siguiente función puede valer.

```
MediaDesviacion = function(x){
  resultado = c(mean(x),sd(x))
  resultado
}
```

La función devuelve lo que tiene en la última línea.[56] Una vez le declaramos la función podemos usarla.[57] La aplicamos a datos generados con distribución normal.

[56] También podemos usar *return*.
[57] Copiamos y pegamos en línea de comandos.

```
x = rnorm(23,mean=12,sd=1.3)
MediaDesviacion(x)

## [1] 12.274904  1.278401
```

Queremos calcular la media y desviación estándar pero de los datos que están por encima del percentil de orden p. El valor de p será un *argumento* y queremos fijarlo por defecto en 0.5.

```
MediaDesviacionPercentil = function(x,ordenpercentil = 0.5){
  qp = quantile(x,probs=ordenpercentil)
  x0 = x[x >qp]
  resultado = c(mean(x0),sd(x0))
  resultado
}
```

Y la utilizamos.

```
MediaDesviacionPercentil(x)

## [1] 13.3144744  0.8567743
```

Si queremos cambiar el valor por defecto del orden del percentil hacemos

```
MediaDesviacionPercentil(x,ordenpercentil=.7)

## [1] 13.7880029  0.6425422
```

No hemos comprobado que efectivamente es un vector. Además pretendemos que no nos devuelva nada si no lo que recibe no es un vector. En este caso que nos avise.

```
MediaDesviacionPercentilError = function(x,ordenpercentil = 0.5){
  if(!is.vector(x)) return("No es un vector")
  qp = quantile(x,probs=ordenpercentil)
  x0 = x[x >qp]
```

```
  resultado = c(mean(x0),sd(x0))
  resultado
}
```

Vamos a ver qué tal funciona. Definimos una matriz.

```
y = matrix(1:16,ncol=4)
```

Comprobamos que no es un vector.

```
is.vector(y)

## [1] FALSE
```

Y probamos nuestra función.

```
MediaDesviacionPercentilError(y)

## [1] "No es un vector"
```

Funciona y todo. No obstante queda un poco feo que cuando es un vector simplemente nos devuelve otro vector y hemos de saber que es una media lo primero y una desviación lo segundo. Podemos devolver una lista de modo que le demos un nombre a cada cosa.

```
MediaDesviacionPercentilErrorLista = function(x,ordenpercentil = 0.5){
  if(!is.vector(x)) return("No es un vector")
  qp = quantile(x,probs=ordenpercentil)
  x0 = x[x >qp]
  resultado = list(media = mean(x0),de = sd(x0))
  resultado
}
```

Y la probamos.

```
MediaDesviacionPercentilErrorLista(x)

## $media
## [1] 13.31447
##
## $de
## [1] 0.8567743
```

Por ello podemos guardar lo que nos devuelve y luego acceder a cada elemento de la lista.

```
x.des = MediaDesviacionPercentilErrorLista(x)
```

Y tendremos que la media es

```
x.des$media

## [1] 13.31447
```

y la desviación estándar sería

```
x.des$de

## [1] 0.8567743
```

** **Ej. 9** — Dado un vector `x` programar una función que nos devuelva en una lista la media y la varianza del vector. Los elementos de la lista se han de llamar `media` y `varianza`.

** **Ej. 10** — Programa una función que tenga como argumentos un vector `x` y un valor numérico `alpha` entre 0 y 1. Eliminar el `alpha` por uno de los más pequeños y la misma cantidad de los más grandes. De lo que queda ha de devolver, en forma de lista, la media y varianza con los nombres `media_ajustada` y `varianza_ajustada`.

2.8 Sobre Bioconductor

En este curso tan (o más) importante que el propio R es Bioconductor. Básicamente es una colección de paquetes de R para Bioinformática. Se instala con

```
source("http://www.bioconductor.org/biocLite.R")
biocLite()
```

Una vez instalado la parte básica de Bioconductor podemos instalar algún paquete adicional (como *ALL*).

```
source("http://www.bioconductor.org/biocLite.R")
biocLite("ALL")
```

Sobre la elección de repositorios. Podemos elegir el repositorio desde el cual instalamos los paquetes de Bioconductor con

```
chooseBioCmirror()
```

También podemos elegir interactivamente el repositorio con

```
setRepositories()
```

* **Ej. 11** — Instalar los paquetes [67, ggfortify], [45, faraway], [108, multtest] y [33, edgeR].

2.9 Paquetes para tami

Los paquetes que utilizamos en este manual , tanto de CRAN como de Bioconductor, los podemos instalar con el script `http://www.uv.es/ayala/docencia/tami/AllTami.R`. En particular contemplamos varias opciones.

- Si pretendemos una instalación de los paquetes básicos hacemos

```
TIPO = BASICO; source("http://www.uv.es/docencia/tami/AllTami.R")
```

- Una instalación completa [58] se tiene con

```
TIPO = TODO; source("http://www.uv.es/docencia/tami/AllTami.R")
```

[58] Tarda un tiempo y se recomienda una buena conexión a Internet.

Figura 2.5: Un anuncio.

Capítulo 3

Anotación

¿De qué variables hablamos? En este manual trabajamos con medidas de abundancia entendida como expresión de un gen o presencia de una mayor cantidad de proteína. En este contexto cuando hablamos de variables utilizaremos con frecuencia el nombre de características. Estas son nuestras variables. Pero, repetimos, ¿de qué variables hablamos? En ocasiones nos estaremos refiriendo a expresión de un gen cuantificado via microarrays o RNASeq o alguna otra técnica. En otras ocasiones hablaremos de exones o bien, cuando sea pertinente, de isoformas del gen obtenidas con empalmes alternativos.[59] Por tanto es de gran importancia[60] poder manejar para el organismo que nos ocupe bases de datos que nos permitan conocer cómo denominar a las características: distintos identificadores de genes, exones, isoformas, proteínas. Además cómo pasar de unos identificadores a otros. Cuáles son las correspondencias entre ellos que con frecuencia no son correspondencias 1-1. Para cada organismo podemos encontrar bases de datos disponibles online en la red. Algunas de ellas se ocupan de más de un organismo. Hay bases de datos específicas de cierto tipo de genes (como de microRNAs). Es habitual en la práctica del investigador acudir y usar este tipo de bases de datos con el gran hallazgo informático de copiar y pegar. Es un trabajo tedioso y, posiblemente, innnesario. ¿Todo lo que hacemos en este tema se puede hacer de este modo? Diría que no. En este tema nos ocupamos de cómo acceder a esta información pero desde R/Bioconductor.[61]

[59] Alternative splicing.

[60] Estamos al principio del texto.

[61] El navegador para leer la prensa.

Es muy conveniente consultar http://www.bioconductor.org/help/workflows/annotation/annotation/. Los paquetes de anotación de Bioconductor los tenemos en https://www.bioconductor.org/packages/release/data/annotation/.

Tipos de paquetes de anotación. Tenemos diferentes tipos de paquetes de anotación:

ChipDb Se refieren a una plataforma concreta. Por ejemplo, un chip de microarray. § 3.2

OrgDb Centrados en el organismo.

TxDb Paquetes relativos a transcriptomas de un organismo.

BSgenome Paquetes con genomas.

[62]

[62] En lo que sigue seguimos el flujo de trabajo annotation de Bioconductor.

Bases de datos e identificadores de genes.

3.1 AnnotationDbi

Las bases de datos de tipo `ChipDb`, `OrgDb` y `TxDb` heredan todos los métodos de la clase `AnnotationDb` que está definida en [105, AnnotationDbi]. Por ello los métodos aplicables a `AnnotationDb` son aplicables a las demás: `columns`, `keytypes`, `keys` y `select`. Con estos métodos podremos extraer información de las bases de las datos (objetos de clase `ChipDb`, `OrgDb` y `TxDb`) correspondientes. Previo al uso de las distintas bases de datos de anotación es pues conveniente conocer los objetos `AnnotationDb`.

```
pacman::p_load("AnnotationDbi")
```

Para ilustrar los métodos vamos a considerar un paquete de tipo `ChipDb`, en concreto [26, hgu133a.db].

```
pacman::p_load("hgu133a.db")
```

La información contenida la podemos ver con

```
ls("package:hgu133a.db")
```

```
##  [1] "hgu133a"              "hgu133a_dbconn"
##  [3] "hgu133a_dbfile"       "hgu133a_dbInfo"
##  [5] "hgu133a_dbschema"     "hgu133a.db"
##  [7] "hgu133aACCNUM"        "hgu133aALIAS2PROBE"
##  [9] "hgu133aCHR"           "hgu133aCHRLENGTHS"
## [11] "hgu133aCHRLOC"        "hgu133aCHRLOCEND"
## [13] "hgu133aENSEMBL"       "hgu133aENSEMBL2PROBE"
## [15] "hgu133aENTREZID"      "hgu133aENZYME"
## [17] "hgu133aENZYME2PROBE"  "hgu133aGENENAME"
## [19] "hgu133aGO"            "hgu133aGO2ALLPROBES"
## [21] "hgu133aGO2PROBE"      "hgu133aMAP"
## [23] "hgu133aMAPCOUNTS"     "hgu133aOMIM"
## [25] "hgu133aORGANISM"      "hgu133aORGPKG"
## [27] "hgu133aPATH"          "hgu133aPATH2PROBE"
## [29] "hgu133aPFAM"          "hgu133aPMID"
## [31] "hgu133aPMID2PROBE"    "hgu133aPROSITE"
## [33] "hgu133aREFSEQ"        "hgu133aSYMBOL"
## [35] "hgu133aUNIGENE"       "hgu133aUNIPROT"
```

Con el nombre del paquete también tenemos información.

```
hgu133a.db
```

```
## ChipDb object:
## | DBSCHEMAVERSION: 2.1
## | Db type: ChipDb
## | Supporting package: AnnotationDbi
## | DBSCHEMA: HUMANCHIP_DB
## | ORGANISM: Homo sapiens
## | SPECIES: Human
```

```
## | MANUFACTURER: Affymetrix
## | CHIPNAME: Human Genome U133 Set
## | MANUFACTURERURL: http://www.affymetrix.com/support/technical/byproduct.affx?produc
## | EGSOURCEDATE: 2015-Sep27
## | EGSOURCENAME: Entrez Gene
## | EGSOURCEURL: ftp://ftp.ncbi.nlm.nih.gov/gene/DATA
## | CENTRALID: ENTREZID
## | TAXID: 9606
## | GOSOURCENAME: Gene Ontology
## | GOSOURCEURL: ftp://ftp.geneontology.org/pub/go/godatabase/archive/latest-lite/
## | GOSOURCEDATE: 20150919
## | GOEGSOURCEDATE: 2015-Sep27
## | GOEGSOURCENAME: Entrez Gene
## | GOEGSOURCEURL: ftp://ftp.ncbi.nlm.nih.gov/gene/DATA
## | KEGGSOURCENAME: KEGG GENOME
## | KEGGSOURCEURL: ftp://ftp.genome.jp/pub/kegg/genomes
## | KEGGSOURCEDATE: 2011-Mar15
## | GPSOURCENAME: UCSC Genome Bioinformatics (Homo sapiens)
## | GPSOURCEURL: ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19
## | GPSOURCEDATE: 2010-Mar22
## | ENSOURCEDATE: 2015-Jul16
## | ENSOURCENAME: Ensembl
## | ENSOURCEURL: ftp://ftp.ensembl.org/pub/current_fasta
## | UPSOURCENAME: Uniprot
## | UPSOURCEURL: http://www.uniprot.org/
## | UPSOURCEDATE: Thu Oct  1 23:31:58 2015
```

En los paquetes de anotación tenemos `columns`. Algunas de estas columnas pueden ser `keys`. Podemos realizar consultas en la base de datos utilizando una `key` y pedir que nos devuelva una o más de una `columns`.

¿Qué información podemos recuperar utilizando `select`?

```
columns(hgu133a.db)
```

```
##  [1] "ACCNUM"       "ALIAS"        "ENSEMBL"
##  [4] "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTREZID"
##  [7] "ENZYME"       "EVIDENCE"     "EVIDENCEALL"
## [10] "GENENAME"     "GO"           "GOALL"
## [13] "IPI"          "MAP"          "OMIM"
## [16] "ONTOLOGY"     "ONTOLOGYALL"  "PATH"
## [19] "PFAM"         "PMID"         "PROBEID"
## [22] "PROSITE"      "REFSEQ"       "SYMBOL"
## [25] "UCSCKG"       "UNIGENE"      "UNIPROT"
```

Pero: ¿qué información es? Lo obtenemos con[63]

```
help("ENTREZID")
```

[63] Podemos poner cualquiera de los nombres anteriores y nos saldrá la misma ayuda.

No todas las variables que hemos obtenido con `columns` son utilizables para realizar consultas. Aquellas utilizables para las consultas las podemos conocer con `keytypes`. A estas variables las llamamos llaves (`keys`).

```
keytypes(hgu133a.db)

##  [1] "ACCNUM"       "ALIAS"        "ENSEMBL"
##  [4] "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTREZID"
##  [7] "ENZYME"       "EVIDENCE"     "EVIDENCEALL"
## [10] "GENENAME"     "GO"           "GOALL"
## [13] "IPI"          "MAP"          "OMIM"
## [16] "ONTOLOGY"     "ONTOLOGYALL"  "PATH"
## [19] "PFAM"         "PMID"         "PROBEID"
## [22] "PROSITE"      "REFSEQ"       "SYMBOL"
## [25] "UCSCKG"       "UNIGENE"      "UNIPROT"
```

¿Cómo conseguir todas los valores de una llave determinada?

```
head(keys(hgu133a.db,keytype="ENTREZID"))

## [1] "10"        "100"       "1000"
## [4] "10000"     "100008586" "10001"

head(keys(hgu133a.db,keytype="ENSEMBL"))

## [1] "ENSG00000121410" "ENSG00000175899"
## [3] "ENSG00000256069" "ENSG00000171428"
## [5] "ENSG00000156006" "ENSG00000196136"
```

Supongamos que tenemos algunos identificadores Affy (AffyID) y pretendemos conocer sus identificadores `ENTREZID`. Empezamos eligiendo cinco identificadores Affy al azar.[64]

[64] De ahí el uso de la función base::sample.

```
(ids = sample(keys(hgu133a.db,keytype="PROBEID"),5))

## [1] "205812_s_at" "220444_at"   "222110_at"
## [4] "214052_x_at" "205021_s_at"
```

Vamos a obtener sus identificadores Entrez, los de ENSEMBL y su SYMBOL.

```
AnnotationDbi::select(hgu133a.db,keys=ids,
                      columns=c("ENTREZID","ENSEMBL","SYMBOL"),
                      keytype="PROBEID")

##       PROBEID ENTREZID         ENSEMBL  SYMBOL
## 1 205812_s_at    27233 ENSG00000198075 SULT1C4
## 2 205812_s_at    54732 ENSG00000184840   TMED9
## 3   220444_at    79230 ENSG00000130544  ZNF557
## 4   222110_at   205564 ENSG00000119231   SENP5
## 5 214052_x_at    23215 ENSG00000117523  PRRC2C
## 6 205021_s_at     1112 ENSG00000053254   FOXN3
```

3.2 ChipDb

Si trabajamos con microarrays (§ 4) nuestras características serán las sondas.[65] Estas sondas tendrán un identificador que el fabricante del chip le ha asignado. Esto no es informativo para nosotros. Hemos

[65] Asociadas a genes.

de poder hacer corresponder este identificador con el gen al que corresponde. Estas bases de datos son de tipo `ChipDb`. Uno de los chips más populares de Affymetrix, Affymetrix Human Genome U133. El paquete Bioconductor con la anotación de este chip es [26, hgu133a.db]. Lo cargamos.

```
pacman::p_load(hgu133a.db)
```

¿Qué sondas[66] tienen correspondencia en Entrez?

[66] Probes.

```
mappedProbes = mappedkeys(hgu133aENTREZID)
```

Lo guardamos en forma de lista.

```
mappedProbesList = as.list(hgu133aENTREZID[mappedProbes])
```

Por ejemplo, la primera posición de la lista nos da el identificador de Affymetrix[67] y su identificador Entrez[68].

[67] AffyID.

[68] ENTREZID.

```
mappedProbesList[1]

## $`1053_at`
## [1] "5982"
```

O el correspondiente a la posición 4567.

```
mappedProbesList[4567]

## $`205255_x_at`
## [1] "6932"
```

Si hacemos

```
ls("package:hgu133a.db")

##  [1] "hgu133a"              "hgu133a_dbconn"
##  [3] "hgu133a_dbfile"       "hgu133a_dbInfo"
##  [5] "hgu133a_dbschema"     "hgu133a.db"
##  [7] "hgu133aACCNUM"        "hgu133aALIAS2PROBE"
##  [9] "hgu133aCHR"           "hgu133aCHRLENGTHS"
## [11] "hgu133aCHRLOC"        "hgu133aCHRLOCEND"
## [13] "hgu133aENSEMBL"       "hgu133aENSEMBL2PROBE"
## [15] "hgu133aENTREZID"      "hgu133aENZYME"
## [17] "hgu133aENZYME2PROBE"  "hgu133aGENENAME"
## [19] "hgu133aGO"            "hgu133aGO2ALLPROBES"
## [21] "hgu133aGO2PROBE"      "hgu133aMAP"
## [23] "hgu133aMAPCOUNTS"     "hgu133aOMIM"
## [25] "hgu133aORGANISM"      "hgu133aORGPKG"
## [27] "hgu133aPATH"          "hgu133aPATH2PROBE"
## [29] "hgu133aPFAM"          "hgu133aPMID"
## [31] "hgu133aPMID2PROBE"    "hgu133aPROSITE"
## [33] "hgu133aREFSEQ"        "hgu133aSYMBOL"
## [35] "hgu133aUNIGENE"       "hgu133aUNIPROT"
```

podemos ver todas las correspondencias que nos ofrece el paquete. Consideremos unos identificadores Affymetrix.

```
ids = c("39730_at", "1635_at", "1674_at", "40504_at", "40202_at")
```

Se supone que el chip es `hgu95av2`. Cargamos el paquete de anotación correspondiente al chip utilizado [27, hgu95av2.db].

```
pacman::p_load("hgu95av2.db")
```

Y lo mismo que antes. Ahora tenemos además la correspondencia entre las sondas y los genes.

```
columns(hgu95av2.db)
```

```
##  [1] "ACCNUM"       "ALIAS"        "ENSEMBL"
##  [4] "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTREZID"
##  [7] "ENZYME"       "EVIDENCE"     "EVIDENCEALL"
## [10] "GENENAME"     "GO"           "GOALL"
## [13] "IPI"          "MAP"          "OMIM"
## [16] "ONTOLOGY"     "ONTOLOGYALL"  "PATH"
## [19] "PFAM"         "PMID"         "PROBEID"
## [22] "PROSITE"      "REFSEQ"       "SYMBOL"
## [25] "UCSCKG"       "UNIGENE"      "UNIPROT"
```

```
keytypes(hgu95av2.db)
```

```
##  [1] "ACCNUM"       "ALIAS"        "ENSEMBL"
##  [4] "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTREZID"
##  [7] "ENZYME"       "EVIDENCE"     "EVIDENCEALL"
## [10] "GENENAME"     "GO"           "GOALL"
## [13] "IPI"          "MAP"          "OMIM"
## [16] "ONTOLOGY"     "ONTOLOGYALL"  "PATH"
## [19] "PFAM"         "PMID"         "PROBEID"
## [22] "PROSITE"      "REFSEQ"       "SYMBOL"
## [25] "UCSCKG"       "UNIGENE"      "UNIPROT"
```

Figura 3.1: Un anuncio.

Los identificadores que teníamos eran los AffyID, esto es, los identificadores de las sondas utilizadas o PROBEID. Podemos plantearnos su correspondencia con el símbolo o nombre del gen y las proteínas asociadas en la base de datos PFAM.

```
columns = c("PFAM","SYMBOL")
AnnotationDbi::select(hgu95av2.db, keys=ids, columns, keytype="PROBEID")
```

3.3 OrgDb

Este tipo de paquete se refiere a un organismo dado y está centrado en el gen. Veamos como ejemplo el relativo a ser humano.

```
pacman::p_load(org.Hs.eg.db)
```

¿Qué tipo de cosas o qué claves podemos manejar?

```
columns(org.Hs.eg.db)
```

```
##  [1] "ACCNUM"       "ALIAS"        "ENSEMBL"
##  [4] "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTREZID"
##  [7] "ENZYME"       "EVIDENCE"     "EVIDENCEALL"
## [10] "GENENAME"     "GO"           "GOALL"
## [13] "IPI"          "MAP"          "OMIM"
## [16] "ONTOLOGY"     "ONTOLOGYALL"  "PATH"
## [19] "PFAM"         "PMID"         "PROSITE"
## [22] "REFSEQ"       "SYMBOL"       "UCSCKG"
## [25] "UNIGENE"      "UNIPROT"
```

O bien con

```
keytypes(org.Hs.eg.db)
```

```
##  [1] "ACCNUM"       "ALIAS"        "ENSEMBL"
##  [4] "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTREZID"
##  [7] "ENZYME"       "EVIDENCE"     "EVIDENCEALL"
## [10] "GENENAME"     "GO"           "GOALL"
## [13] "IPI"          "MAP"          "OMIM"
## [16] "ONTOLOGY"     "ONTOLOGYALL"  "PATH"
## [19] "PFAM"         "PMID"         "PROSITE"
## [22] "REFSEQ"       "SYMBOL"       "UCSCKG"
## [25] "UNIGENE"      "UNIPROT"
```

Por ejemplo, si queremos los primeros identificadores de genes utilizando los identificadores Entrez o ENTREZID tenemos

```
head(keys(org.Hs.eg.db, keytype="ENTREZID"))
```

```
## [1] "1"  "2"  "3"  "9"  "10" "11"
```

Si consideramos sus identificadores en la base de datos Ensembl entonces podemos usar

```
head(keys(org.Hs.eg.db, keytype="ENSEMBL"))
```

```
## [1] "ENSG00000121410" "ENSG00000175899"
## [3] "ENSG00000256069" "ENSG00000171428"
## [5] "ENSG00000156006" "ENSG00000196136"
```

Y en Gene Ontology.

```
head(keys(org.Hs.eg.db, keytype="GO"))
```

```
## [1] "GO:0002576" "GO:0003674" "GO:0005576"
## [4] "GO:0005615" "GO:0008150" "GO:0031093"
```

Supongamos que elegimos un sistema de identificación, por ejemplo, ENTREZID. En concreto, los cinco primeros genes.

```
(ids = keys(org.Hs.eg.db, keytype="ENTREZID")[1:5])
```

```
## [1] "1"  "2"  "3"  "9"  "10"
```

A partir de estos identicadores podemos obtener el resto.

```
AnnotationDbi::select(org.Hs.eg.db,keys=ids,column="SYMBOL",keytype='ENTREZID')
```

```
##   ENTREZID SYMBOL
## 1        1   A1BG
## 2        2    A2M
## 3        3  A2MP1
## 4        9   NAT1
## 5       10   NAT2
```

Supongamos que nos fijamos en el gen con código Ensembl `ENSG00000000003`.

```
(id = "ENSG00000171428")
```

```
## [1] "ENSG00000171428"
```

Buscamos su correspondencia en Gene Ontology.

```
(res =  AnnotationDbi::select(org.Hs.eg.db, keys=id, column="GO", keytype="ENSEMBL"))
```

```
##           ENSEMBL         GO EVIDENCE ONTOLOGY
## 1 ENSG00000171428 GO:0004060      IBA       MF
## 2 ENSG00000171428 GO:0004060      TAS       MF
## 3 ENSG00000171428 GO:0005829      TAS       CC
## 4 ENSG00000171428 GO:0006805      TAS       BP
```

Como vemos no tenemos una correspondencia 1-1. Al mismo gen le corresponden distintos términos Gene Ontology. Si solamente tenemos interés en ellos podemos hacer

```
res[,"GO"]
```

```
## [1] "GO:0004060" "GO:0004060" "GO:0005829"
## [4] "GO:0006805"
```

Puesto que tenemos identificadores Gene Ontology podemos utilizar el paquete [25, GO.db] para obtener los términos Gene Ontology correspondientes.

```
pacman::p_load(GO.db)
```

Y los términos Gene Ontology serían

```
AnnotationDbi::select(GO.db, keys=res[,"GO"], columns="TERM", keytype="GOID")
```

3.4 TxDb

Los paquetes TxDb están centrados en el genoma. Vamos a trabajar con la mosca de la drosophila melanogaster.[69] Cargamos la base de datos.

[69] https://en.wikipedia.org/wiki/Drosophila_melanogaster.

```
library("GenomicFeatures")
library("TxDb.Dmelanogaster.UCSC.dm3.ensGene")
```

Al cargar este paquete lo que hemos hecho es leer un objeto que se llama como el propio paquete y de clase `TxDb`.[70]

[70] Todos los paquetes de Bioconductor que empiezan con TxDb son de este tipo.

```
class(TxDb.Dmelanogaster.UCSC.dm3.ensGene)

## [1] "TxDb"
## attr(,"package")
## [1] "GenomicFeatures"
```

Hacemos una copia con un nombre más breve.

```
txdb = TxDb.Dmelanogaster.UCSC.dm3.ensGene
```

¿De qué clase es este objeto?

```
class(txdb)

## [1] "TxDb"
## attr(,"package")
## [1] "GenomicFeatures"
```

Es pues un objeto de clase `TxDb`. Tenemos un resumen sobre los datos contenidos en `txdb` con

```
txdb

## TxDb object:
## # Db type: TxDb
## # Supporting package: GenomicFeatures
## # Data source: UCSC
## # Genome: dm3
## # Organism: Drosophila melanogaster
## # Taxonomy ID: 7227
## # UCSC Table: ensGene
## # Resource URL: http://genome.ucsc.edu/
## # Type of Gene ID: Ensembl gene ID
## # Full dataset: yes
## # miRBase build ID: NA
## # transcript_nrow: 29173
## # exon_nrow: 76920
## # cds_nrow: 62135
## # Db created by: GenomicFeatures package from Bioconductor
## # Creation time: 2015-10-07 18:15:53 +0000 (Wed, 07 Oct 2015)
## # GenomicFeatures version at creation time: 1.21.30
## # RSQLite version at creation time: 1.0.0
## # DBSCHEMAVERSION: 1.1
```

Podemos ver la información de la que disponemos con

```
columns(txdb)
```

```
##  [1] "CDSCHROM"   "CDSEND"     "CDSID"
##  [4] "CDSNAME"    "CDSSTART"   "CDSSTRAND"
##  [7] "EXONCHROM"  "EXONEND"    "EXONID"
## [10] "EXONNAME"   "EXONRANK"   "EXONSTART"
## [13] "EXONSTRAND" "GENEID"     "TXCHROM"
## [16] "TXEND"      "TXID"       "TXNAME"
## [19] "TXSTART"    "TXSTRAND"   "TXTYPE"
```

y con

```
keytypes(txdb)

## [1] "CDSID"    "CDSNAME"  "EXONID"   "EXONNAME"
## [5] "GENEID"   "TXID"     "TXNAME"
```

Para el manejo de este tipo de bases de datos es útil [23, GenomicFeatures].

```
pacman::p_load(GenomicFeatures)
```

Por ejemplo: ¿Qué cromosomas tenemos?[71]

[71] En http://ucscbrowser.genenetwork.org/cgi-bin/hgGateway?hgsid=732&clade=insect&org=0&db=0 podemos encontrar una explicación detallada.

```
seqlevels(txdb)

## [1] "chr2L"
```

Cuando se carga la base de datos todos los cromosomas están activos y lo que hagamos nos dará información sobre todos ellos. Supongamos que no queremos esto. Queremos, por ejemplo, trabajar solamente con `chr2L`. Lo conseguimos con

```
seqlevels(txdb) = "chr2L"
```

Supongamos que queremos conocer los `GENEID` de los primeros genes.

```
(keysGENEID = head(keys(txdb, keytype="GENEID"),n=3))

## [1] "FBgn0000003" "FBgn0000008" "FBgn0000014"
```

```
columns = c("TXNAME", "TXSTART","TXEND","TXSTRAND")
AnnotationDbi::select(txdb, keysGENEID, columns, keytype="GENEID")

##        GENEID      TXNAME TXSTRAND  TXSTART
## 1 FBgn0000003 FBtr0081624        +  2648220
## 2 FBgn0000008 FBtr0100521        + 18024494
## 3 FBgn0000008 FBtr0071763        + 18024496
## 4 FBgn0000008 FBtr0071764        + 18024938
## 5 FBgn0000014 FBtr0306337        - 12632936
## 6 FBgn0000014 FBtr0083388        - 12633349
## 7 FBgn0000014 FBtr0083387        - 12633349
## 8 FBgn0000014 FBtr0300485        - 12633349
##      TXEND
## 1  2648518
```

```
## 2 18060339
## 3 18060346
## 4 18060346
## 5 12655767
## 6 12653845
## 7 12655300
## 8 12655474
```

Nos devuelve el identificador del gen (`GENEID`), el nombre del transcrito, la hebra o cadena y el punto de inicio y de finalización. Notemos que el primer gen solo tiene un transcrito mientras que el segundo gen tiene tres transcritos.

Los objetos `TxDb` nos permiten obtener las anotaciones como `GRanges`. Empezamos con los transcritos.

```
(txdb.tr = transcripts(txdb))

## GRanges object with 5384 ranges and 2 metadata columns:
##          seqnames            ranges strand |
##             <Rle>         <IRanges>  <Rle> |
##      [1]    chr2L         7529-9484      + |
##      [2]    chr2L         7529-9484      + |
##      [3]    chr2L         7529-9484      + |
##      [4]    chr2L       21952-24237      + |
##      [5]    chr2L       66584-71390      + |
##      ...      ...               ...    ... .
##   [5380]    chr2L 22892306-22918560      - |
##   [5381]    chr2L 22892306-22918647      - |
##   [5382]    chr2L 22959606-22960915      - |
##   [5383]    chr2L 22959606-22961179      - |
##   [5384]    chr2L 22959606-22961179      - |
##              tx_id     tx_name
##          <integer> <character>
##      [1]         1 FBtr0300689
##      [2]         2 FBtr0300690
##      [3]         3 FBtr0330654
##      [4]         4 FBtr0309810
##      [5]         5 FBtr0306539
##      ...       ...         ...
##   [5380]      5380 FBtr0331166
##   [5381]      5381 FBtr0111127
##   [5382]      5382 FBtr0111241
##   [5383]      5383 FBtr0111239
##   [5384]      5384 FBtr0111240
##   -------
##   seqinfo: 1 sequence from dm3 genome
```

Como estamos trabajando con el cromosoma `chr2L` nos devuelve la información en este nuevo (y mejor) formato. Los que ocupan las posiciones 1, 2, 3 y 1000 serían

```
txdb.tr[c(1:3,1000)]

## GRanges object with 4 ranges and 2 metadata columns:
```

```
##       seqnames          ranges strand |     tx_id
##          <Rle>       <IRanges>  <Rle> | <integer>
##   [1]    chr2L       7529-9484      + |         1
##   [2]    chr2L       7529-9484      + |         2
##   [3]    chr2L       7529-9484      + |         3
##   [4]    chr2L 8011405-8026898      + |      1000
##           tx_name
##       <character>
##   [1] FBtr0300689
##   [2] FBtr0300690
##   [3] FBtr0330654
##   [4] FBtr0079533
##   -------
##   seqinfo: 1 sequence from dm3 genome
```

También podemos obtener información sobre los exones en modo de un objeto `GRanges`.

```
(txdb.ex = exons(txdb))
```

```
## GRanges object with 13850 ranges and 1 metadata column:
##           seqnames            ranges strand |
##              <Rle>         <IRanges>  <Rle> |
##       [1]    chr2L         7529-8116      + |
##       [2]    chr2L         8193-8589      + |
##       [3]    chr2L         8193-9484      + |
##       [4]    chr2L         8229-9484      + |
##       [5]    chr2L         8668-9484      + |
##       ...      ...               ...    ... .
##   [13846]    chr2L 22959606-22959815      - |
##   [13847]    chr2L 22959877-22960833      - |
##   [13848]    chr2L 22959877-22960876      - |
##   [13849]    chr2L 22959877-22960915      - |
##   [13850]    chr2L 22960932-22961179      - |
##             exon_id
##           <integer>
##       [1]         1
##       [2]         2
##       [3]         3
##       [4]         4
##       [5]         5
##       ...       ...
##   [13846]     13846
##   [13847]     13847
##   [13848]     13848
##   [13849]     13849
##   [13850]     13850
##   -------
##   seqinfo: 1 sequence from dm3 genome
```

Como antes el exon que ocupa la posición 123 sería

```
txdb.ex[123]
```

```
## GRanges object with 1 range and 1 metadata column:
```

```
##       seqnames        ranges strand |   exon_id
##          <Rle>     <IRanges>  <Rle> | <integer>
##   [1]    chr2L 277930-278323      + |       123
##   -------
##   seqinfo: 1 sequence from dm3 genome
```

Podemos incluir metadatos adicionales como puede ser el identificador del gen.

```
transcripts(txdb, columns = c("tx_id","tx_name","gene_id"))
```

```
## GRanges object with 5384 ranges and 3 metadata columns:
##          seqnames            ranges strand |
##             <Rle>         <IRanges>  <Rle> |
##      [1]    chr2L         7529-9484      + |
##      [2]    chr2L         7529-9484      + |
##      [3]    chr2L         7529-9484      + |
##      [4]    chr2L       21952-24237      + |
##      [5]    chr2L       66584-71390      + |
##      ...      ...               ...    ... .
##   [5380]    chr2L 22892306-22918560      - |
##   [5381]    chr2L 22892306-22918647      - |
##   [5382]    chr2L 22959606-22960915      - |
##   [5383]    chr2L 22959606-22961179      - |
##   [5384]    chr2L 22959606-22961179      - |
##              tx_id     tx_name         gene_id
##          <integer> <character> <CharacterList>
##      [1]         1 FBtr0300689     FBgn0031208
##      [2]         2 FBtr0300690     FBgn0031208
##      [3]         3 FBtr0330654     FBgn0031208
##      [4]         4 FBtr0309810     FBgn0263584
##      [5]         5 FBtr0306539     FBgn0067779
##      ...       ...         ...             ...
##   [5380]      5380 FBtr0331166     FBgn0250907
##   [5381]      5381 FBtr0111127     FBgn0250907
##   [5382]      5382 FBtr0111241     FBgn0086683
##   [5383]      5383 FBtr0111239     FBgn0086683
##   [5384]      5384 FBtr0111240     FBgn0086683
##   -------
##   seqinfo: 1 sequence from dm3 genome
```

Obtenemos las regiones CDS con

```
(txdb.cds = cds(txdb))
```

```
## GRanges object with 11003 ranges and 1 metadata column:
##           seqnames            ranges strand |
##              <Rle>         <IRanges>  <Rle> |
##       [1]    chr2L         7680-8116      + |
##       [2]    chr2L         8193-8589      + |
##       [3]    chr2L         8193-8610      + |
##       [4]    chr2L         8229-8610      + |
##       [5]    chr2L         8668-9276      + |
##       ...      ...               ...    ... .
```

```
##   [10999]    chr2L 22959877-22960833      - |
##   [11000]    chr2L 22959877-22960873      - |
##   [11001]    chr2L 22959877-22960876      - |
##   [11002]    chr2L 22960932-22960995      - |
##   [11003]    chr2L 22960932-22961048      - |
##              cds_id
##           <integer>
##       [1]         1
##       [2]         2
##       [3]         3
##       [4]         4
##       [5]         5
##       ...       ...
##   [10999]     10999
##   [11000]     11000
##   [11001]     11001
##   [11002]     11002
##   [11003]     11003
##   -------
##   seqinfo: 1 sequence from dm3 genome
```

En los tres casos hemos obtenido un objeto `GRanges`. A este tipo de objetos podemos aplicar otros métodos que nos dan información adicional. Por ejemplo, el número de elementos que lo componen

```
length(txdb.cds)

## [1] 11003
```

Podemos ver la hebra en la que están[72]

[72] En la salida que sigue nos hace una tabla de frecuencias.

```
strand(txdb.cds)

## factor-Rle of length 11003 with 2 runs
##   Lengths: 5489 5514
##   Values :    +    -
## Levels(3): + - *
```

A partir de un objeto de clase `TxDb` podemos obtener un `GRangesList` en la cual separamos por alguna característica. Por ejemplo, los objetos `GRanges` para los distintos genes.

```
transcriptsBy(txdb, by="gene")

## GRangesList object of length 2986:
## $FBgn0000018
## GRanges object with 1 range and 2 metadata columns:
##       seqnames            ranges strand |
##          <Rle>         <IRanges>  <Rle> |
##   [1]    chr2L 10973443-10975273      - |
##           tx_id     tx_name
##       <integer> <character>
##   [1]      4279 FBtr0080168
##   -------
##   seqinfo: 1 sequence from dm3 genome
```

```
##
## $FBgn0000052
## GRanges object with 3 ranges and 2 metadata columns:
##       seqnames          ranges strand |     tx_id
##          <Rle>       <IRanges>  <Rle> | <integer>
##   [1]    chr2L 6041178-6045593      - |      3537
##   [2]    chr2L 6041178-6045970      - |      3538
##   [3]    chr2L 6041178-6045970      - |      3539
##           tx_name
##       <character>
##   [1] FBtr0079221
##   [2] FBtr0079219
##   [3] FBtr0079220
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## $FBgn0000053
## GRanges object with 2 ranges and 2 metadata columns:
##       seqnames          ranges strand |     tx_id
##          <Rle>       <IRanges>  <Rle> | <integer>
##   [1]    chr2L 7014861-7023940      - |      3704
##   [2]    chr2L 7018085-7023940      - |      3705
##           tx_name
##       <character>
##   [1] FBtr0079431
##   [2] FBtr0100353
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## ...
## <2983 more elements>
```

Podemos agrupar los exones por gen.

```
exonsBy(txdb, by="gene")
```

```
## GRangesList object of length 2986:
## $FBgn0000018
## GRanges object with 2 ranges and 2 metadata columns:
##       seqnames            ranges strand |
##          <Rle>         <IRanges>  <Rle> |
##   [1]    chr2L 10973443-10974210      - |
##   [2]    chr2L 10974268-10975273      - |
##         exon_id   exon_name
##       <integer> <character>
##   [1]     11021        <NA>
##   [2]     11022        <NA>
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## $FBgn0000052
## GRanges object with 6 ranges and 2 metadata columns:
##       seqnames          ranges strand |   exon_id
##          <Rle>       <IRanges>  <Rle> | <integer>
```

```
##   [1]    chr2L 6041178-6042011      - |      8949
##   [2]    chr2L 6042075-6044351      - |      8950
##   [3]    chr2L 6044428-6045526      - |      8951
##   [4]    chr2L 6044428-6045593      - |      8952
##   [5]    chr2L 6045894-6045970      - |      8955
##   [6]    chr2L 6045921-6045970      - |      8956
##         exon_name
##       <character>
##   [1]        <NA>
##   [2]        <NA>
##   [3]        <NA>
##   [4]        <NA>
##   [5]        <NA>
##   [6]        <NA>
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## $FBgn0000053
## GRanges object with 8 ranges and 2 metadata columns:
##       seqnames          ranges strand |   exon_id
##          <Rle>       <IRanges>  <Rle> | <integer>
##   [1]    chr2L 7014861-7016374      - |      9389
##   [2]    chr2L 7016437-7017486      - |      9390
##   [3]    chr2L 7017545-7017966      - |      9391
##   [4]    chr2L 7018085-7018374      - |      9392
##   [5]    chr2L 7018149-7018374      - |      9393
##   [6]    chr2L 7018431-7019080      - |      9394
##   [7]    chr2L 7019135-7019382      - |      9395
##   [8]    chr2L 7023529-7023940      - |      9396
##         exon_name
##       <character>
##   [1]        <NA>
##   [2]        <NA>
##   [3]        <NA>
##   [4]        <NA>
##   [5]        <NA>
##   [6]        <NA>
##   [7]        <NA>
##   [8]        <NA>
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## ...
## <2983 more elements>
```

O bien podemos tener los CDS agrupados por transcrito.

```
cdsBy(txdb, by="tx")
```

```
## GRangesList object of length 4951:
## $`1`
## GRanges object with 2 ranges and 3 metadata columns:
##       seqnames    ranges strand |    cds_id
##          <Rle> <IRanges>  <Rle> | <integer>
```

```
##   [1]    chr2L 7680-8116      + |         1
##   [2]    chr2L 8193-8610      + |         3
##          cds_name exon_rank
##       <character> <integer>
##   [1]        <NA>         1
##   [2]        <NA>         2
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## $`2`
## GRanges object with 3 ranges and 3 metadata columns:
##       seqnames    ranges strand |    cds_id
##          <Rle> <IRanges>  <Rle> | <integer>
##   [1]    chr2L 7680-8116      + |         1
##   [2]    chr2L 8193-8589      + |         2
##   [3]    chr2L 8668-9276      + |         5
##          cds_name exon_rank
##       <character> <integer>
##   [1]        <NA>         1
##   [2]        <NA>         2
##   [3]        <NA>         3
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## $`3`
## GRanges object with 2 ranges and 3 metadata columns:
##       seqnames    ranges strand |    cds_id
##          <Rle> <IRanges>  <Rle> | <integer>
##   [1]    chr2L 7680-8116      + |         1
##   [2]    chr2L 8229-8610      + |         4
##          cds_name exon_rank
##       <character> <integer>
##   [1]        <NA>         1
##   [2]        <NA>         2
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## ...
## <4948 more elements>
```

O los intrones agrupados por transcrito.

```
intronsByTranscript(txdb)
```

```
## GRangesList object of length 5384:
## $`1`
## GRanges object with 1 range and 0 metadata columns:
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]    chr2L 8117-8192      +
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## $`2`
```

```
## GRanges object with 2 ranges and 0 metadata columns:
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]    chr2L 8117-8192      +
##   [2]    chr2L 8590-8667      +
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## $`3`
## GRanges object with 1 range and 0 metadata columns:
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]    chr2L 8117-8228      +
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## ...
## <5381 more elements>
```

También podemos tener las regiones UTR 5' y 3' agrupadas por transcrito.

```
fiveUTRsByTranscript(txdb)
```

```
## GRangesList object of length 4733:
## $`1`
## GRanges object with 1 range and 3 metadata columns:
##       seqnames    ranges strand |   exon_id
##          <Rle> <IRanges>  <Rle> | <integer>
##   [1]    chr2L 7529-7679      + |         1
##         exon_name exon_rank
##       <character> <integer>
##   [1]        <NA>         1
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## $`2`
## GRanges object with 1 range and 3 metadata columns:
##       seqnames    ranges strand |   exon_id
##          <Rle> <IRanges>  <Rle> | <integer>
##   [1]    chr2L 7529-7679      + |         1
##         exon_name exon_rank
##       <character> <integer>
##   [1]        <NA>         1
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## $`3`
## GRanges object with 1 range and 3 metadata columns:
##       seqnames    ranges strand |   exon_id
##          <Rle> <IRanges>  <Rle> | <integer>
##   [1]    chr2L 7529-7679      + |         1
##         exon_name exon_rank
##       <character> <integer>
```

```
##   [1]        <NA>         1
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## ...
## <4730 more elements>
```

```
threeUTRsByTranscript(txdb)
```

```
## GRangesList object of length 4721:
## $`1`
## GRanges object with 1 range and 3 metadata columns:
##       seqnames    ranges strand |   exon_id
##          <Rle> <IRanges>  <Rle> | <integer>
##   [1]    chr2L 8611-9484      + |         3
##         exon_name exon_rank
##       <character> <integer>
##   [1]        <NA>         2
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## $`2`
## GRanges object with 1 range and 3 metadata columns:
##       seqnames    ranges strand |   exon_id
##          <Rle> <IRanges>  <Rle> | <integer>
##   [1]    chr2L 9277-9484      + |         5
##         exon_name exon_rank
##       <character> <integer>
##   [1]        <NA>         3
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## $`3`
## GRanges object with 1 range and 3 metadata columns:
##       seqnames    ranges strand |   exon_id
##          <Rle> <IRanges>  <Rle> | <integer>
##   [1]    chr2L 8611-9484      + |         4
##         exon_name exon_rank
##       <character> <integer>
##   [1]        <NA>         2
##   -------
##   seqinfo: 1 sequence from dm3 genome
##
## ...
## <4718 more elements>
```

3.5 BSgenome

Estos paquetes contienen datos de secuencias para organismos secuenciados.

BSgenome.Dmelanogaster.UCSC.dm3

```
pacman::p_load(BSgenome.Dmelanogaster.UCSC.dm3)
tx2seqs = extractTranscriptSeqs(BSgenome.Dmelanogaster.UCSC.dm3,TxDb.Dmelanogaster.UCSC
```

La secuencia correspondiente al primer gen sería

```
tx2seqs[[1]]

## 1880-letter DNAString object
## seq: CTACTCGCATGTAGAGATTTC...TTCAATAAAATTTCCCCAAGT
```

Si queremos conocer la secuencia entre las posiciones 1000 y 1020 entonces

```
tx2seqs[[1]][1000:1020]

## 21-letter DNAString object
## seq: ATATTGATGTCTTTCGTACCC
```

También podemos trasladar estas secuencias a proteínas con

```
suppressWarnings(translate(tx2seqs[[1]]))
```

Para todos los transcritos lo hacemos con

```
suppressWarnings(translate(tx2seqs))
```

No todo lo que se transcribe se traslada. Para obtener obtener las que realmente se trasladan se puede hacer

```
cds2seqs = extractTranscriptSeqs(BSgenome.Dmelanogaster.UCSC.dm3,
                                 cdsBy(txdb, by="tx"))
translate(cds2seqs)
```

BSgenome.Hsapiens.UCSC.hg19

Estos paquetes contienen datos de secuencias para organismos secuenciados.

```
pacman::p_load(BSgenome.Hsapiens.UCSC.hg19)
```

```
Hsapiens

## Human genome:
## # organism: Homo sapiens (Human)
## # provider: UCSC
## # provider version: hg19
## # release date: June 2013
## # release name: Genome Reference Consortium GRCh37.p13
## # 298 sequences:
## #   chr1                  chr2
## #   chr3                  chr4
## #   chr5                  chr6
```

```
## #   chr7                  chr8
## #   chr9                  chr10
## #   ...                   ...
## #   chr19_gl949752_alt    chr19_gl949753_alt
## #   chr20_gl383577_alt    chr21_gl383578_alt
## #   chr21_gl383579_alt    chr21_gl383580_alt
## #   chr21_gl383581_alt    chr22_gl383582_alt
## #   chr22_gl383583_alt    chr22_kb663609_alt
## # (use 'seqnames()' to see all the sequence names,
## # use the '$' or '[[' operator to access a given
## # sequence)
```

Nos fijamos en las secuencias.

```
seqNms = seqnames(Hsapiens)
head(seqNms)

## [1] "chr1" "chr2" "chr3" "chr4" "chr5" "chr6"
```

Nos fijamos en las dos primeras.

```
getSeq(Hsapiens, seqNms[1:2])

## DNAStringSet object of length 2:
##         width seq             names
## [1] 249250621 NNNNNN...NNNNNN chr1
## [2] 243199373 NNNNNN...NNNNNN chr2
```

Podemos, utilizando un objeto GRanges, obtener la secuencia de bases en los cromosomas que queramos, en la hebra que queramos, entre las posiciones que queramos.

```
rngs <- GRanges(seqnames = c('chr1', 'chr4'), strand=c('+','-'),
                ranges = IRanges(start=c(100000,300000),
                    end=c(100023,300037)))
rngs

## GRanges object with 2 ranges and 0 metadata columns:
##       seqnames        ranges strand
##          <Rle>     <IRanges>  <Rle>
##   [1]     chr1 100000-100023      +
##   [2]     chr4 300000-300037      -
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths

res <- getSeq(Hsapiens, rngs)
res

## DNAStringSet object of length 2:
##     width seq
## [1]    24 CACTAAGCACACAGAGAATAATGT
## [2]    38 GCTGGTCCCTTACTTCCAGTAGAAAAGACGTGTTCAGG
```

3.6 OrganismDb

Un paquete de tipo OrganismDb nos permite combinar información (para el organismo con que estemos trabajando) de GO.db con el correspondiente TxDb y OrgDb.[73]

[73] Buscar OrganismDb en Bioconductor.

```
pacman::p_load(Homo.sapiens)
```

Tomamos las dos primeras.

```
keys = head(keys(Homo.sapiens, keytype="ENTREZID"), n=2)
columns = c("SYMBOL","TXNAME")
AnnotationDbi::select(Homo.sapiens, keys, columns, keytype="ENTREZID")
```

```
##   ENTREZID SYMBOL     TXNAME
## 1        1   A1BG uc002qsd.4
## 2        1   A1BG uc002qsf.2
## 3        2    A2M uc001qvk.1
## 4        2    A2M uc009zgk.1
```

También podemos obtener `GRanges`.

```
transcripts(Homo.sapiens, columns=c("TXNAME","SYMBOL"))
```

```
## GRanges object with 82960 ranges and 2 metadata columns:
##                 seqnames        ranges strand |
##                    <Rle>     <IRanges>  <Rle> |
##       [1]           chr1   11874-14409      + |
##       [2]           chr1   11874-14409      + |
##       [3]           chr1   11874-14409      + |
##       [4]           chr1   69091-70008      + |
##       [5]           chr1 321084-321115      + |
##       ...            ...           ...    ... .
##   [82956] chrUn_gl000237        1-2686      - |
##   [82957] chrUn_gl000241   20433-36875      - |
##   [82958] chrUn_gl000243   11501-11530      + |
##   [82959] chrUn_gl000243   13608-13637      + |
##   [82960] chrUn_gl000247     5787-5816      - |
##                    TXNAME          SYMBOL
##           <CharacterList> <CharacterList>
##       [1]      uc001aaa.3         DDX11L1
##       [2]      uc010nxq.1         DDX11L1
##       [3]      uc010nxr.1         DDX11L1
##       [4]      uc001aal.1           OR4F5
##       [5]      uc001aaq.2              NA
##       ...             ...             ...
##   [82956]      uc011mgu.1              NA
##   [82957]      uc011mgv.2              NA
##   [82958]      uc011mgw.1              NA
##   [82959]      uc022brq.1              NA
##   [82960]      uc022brr.1              NA
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

3.7 biomaRt

El paquete [41, biomaRt] es un interfaz para poder acceder a una serie de bases de datos que implementan BioMart.[74]

[74] http://www.biomart.org.

```
pacman::p_load(biomaRt)
```

Podemos ver la lista de `mart`'s que tenemos (mostramos los primeros).

```
listMarts(host="www.ensembl.org")
```

```
##                biomart                version
## 1 ENSEMBL_MART_ENSEMBL      Ensembl Genes 100
## 2   ENSEMBL_MART_MOUSE      Mouse strains 100
## 3     ENSEMBL_MART_SNP  Ensembl Variation 100
## 4 ENSEMBL_MART_FUNCGEN Ensembl Regulation 100
```

Elegimos utilizar `ensembl`.

```
(ensembl = useMart("ENSEMBL_MART_ENSEMBL",host="www.ensembl.org"))
```

```
## Object of class 'Mart':
##   Using the ENSEMBL_MART_ENSEMBL BioMart database
##   No dataset selected.
```

Ahora hemos de elegir el conjunto de datos a utilizar. Podemos ver los disponibles con

```
head(listDatasets(ensembl))
```

```
##                      dataset
## 1   acalliptera_gene_ensembl
## 2 acarolinensis_gene_ensembl
## 3  acchrysaetos_gene_ensembl
## 4  acitrinellus_gene_ensembl
## 5  amelanoleuca_gene_ensembl
## 6    amexicanus_gene_ensembl
##                                    description
## 1             Eastern happy genes (fAstCal1.2)
## 2               Anole lizard genes (AnoCar2.0)
## 3              Golden eagle genes (bAquChr1.2)
## 4               Midas cichlid genes (Midas_v5)
## 5                        Panda genes (ailMel1)
## 6 Mexican tetra genes (Astyanax_mexicanus-2.0)
##                  version
## 1             fAstCal1.2
## 2              AnoCar2.0
## 3             bAquChr1.2
## 4               Midas_v5
## 5                ailMel1
## 6 Astyanax_mexicanus-2.0
```

Elegimos la correspondiente a ser humano.

```
(ensembl = useMart("ENSEMBL_MART_ENSEMBL",dataset="hsapiens_gene_ensembl",
     host="www.ensembl.org"))
```

Podemos ver los atributos con

```
head(listAttributes(ensembl))
```

De hecho, son

```
nrow(listAttributes(ensembl))
```

Podemos comprobar que hay muchos que identifican el gen con las sondas de Affymetrix, en concreto, con los AffyID. Supongamos que nos fijamos en los siguientes AffyID.

```
affyids=c("202763_at","209310_s_at","207500_at")
```

¿A qué genes corresponden?

```
getBM(attributes=c('affy_hg_u133_plus_2', 'entrezgene'),
                   filters = 'affy_hg_u133_plus_2',
                   values = affyids, mart = ensembl)
```

Podemos obtener todos los identificadores.

```
head(getBM(attributes='affy_hg_u133_plus_2', mart = ensembl))
```

3.8 KEGGREST

Con [131, KEGGREST] podemos acceder a la base de datos KEGG. En concreto permite el acceso a KEGG REST API.

3.9 Tareas habituales con anotaciones

¿Cuáles son las tareas que tenemos que realizar habitualmente? En esta sección vemos posibles soluciones que las resuelvan.

3.9.1 Cambiar identificadores de un ExpressionSet

[75] § 5.10.

Consideremos el ExpressionSet tamidata::gse1397.[75] Las características están codificadas utilizando los identificadores Affymetrix. Pretendemos incorporar la información que nos permita conocer la correspondencia entre estos identificadores (PROBEID) y los identificadores Entrez y Ensembl. Esta información la incorporamos como `fData` del ExpressionSet.

Necesitamos los paquetes.

```
pacman::p_load("Biobase","AnnotationDbi","BiocGenerics")
```

Leemos los datos.

```
data(gse1397,package="tamidata")
```

¿Qué paquete de anotación necesitamos?

```
Biobase::annotation(gse1397)

## [1] "hgu133a"
```

Lo cargamos.[76]

```
pacman::p_load("hgu133a.db")
```

Como hemos visto en § 3.1 con AnnotationDbi::columns podemos ver la información que tenemos y con AnnotationDbi::keytypes las llaves con las que podemos realizar consultas.

```
columns(hgu133a.db)

##  [1] "ACCNUM"       "ALIAS"        "ENSEMBL"
##  [4] "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTREZID"
##  [7] "ENZYME"       "EVIDENCE"     "EVIDENCEALL"
## [10] "GENENAME"     "GO"           "GOALL"
## [13] "IPI"          "MAP"          "OMIM"
## [16] "ONTOLOGY"     "ONTOLOGYALL"  "PATH"
## [19] "PFAM"         "PMID"         "PROBEID"
## [22] "PROSITE"      "REFSEQ"       "SYMBOL"
## [25] "UCSCKG"       "UNIGENE"      "UNIPROT"

keytypes(hgu133a.db)

##  [1] "ACCNUM"       "ALIAS"        "ENSEMBL"
##  [4] "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTREZID"
##  [7] "ENZYME"       "EVIDENCE"     "EVIDENCEALL"
## [10] "GENENAME"     "GO"           "GOALL"
## [13] "IPI"          "MAP"          "OMIM"
## [16] "ONTOLOGY"     "ONTOLOGYALL"  "PATH"
## [19] "PFAM"         "PMID"         "PROBEID"
## [22] "PROSITE"      "REFSEQ"       "SYMBOL"
## [25] "UCSCKG"       "UNIGENE"      "UNIPROT"
```

Vemos que, en este caso coinciden `columns` y `keys`. Los valores los tenemos con

```
head(keys(hgu133a.db,keytype="PROBEID"))

## [1] "1007_s_at" "1053_at"   "117_at"
## [4] "121_at"    "1255_g_at" "1294_at"
```

que corresponde con los identificadores Affy o identificadores de las sondas. Podemos ver los primeros que tenemos en nuestros datos.

```
head(featureNames(gse1397))

## [1] "1007_s_at" "1053_at"   "117_at"
## [4] "121_at"    "1255_g_at" "1294_at"
```

[76] En https://www.bioconductor.org/packages/3.3/data/annotation/ tenemos el listado de los paquetes de anotación de los que dispone Bioconductor.

¿Coinciden todos?

```
table(featureNames(gse1397) == keys(hgu133a.db,keytype="PROBEID"))
```

```
##
## FALSE  TRUE
##    53 22230
```

Vemos que no todos coinciden. ¿Quienes son?

```
(control = which(featureNames(gse1397) != keys(hgu133a.db,keytype="PROBEID")))
```

```
##  [1] 22231 22232 22233 22234 22235 22236 22237
##  [8] 22238 22239 22240 22241 22242 22243 22244
## [15] 22245 22246 22247 22248 22249 22250 22251
## [22] 22252 22253 22254 22255 22256 22257 22258
## [29] 22259 22260 22261 22262 22263 22264 22265
## [36] 22266 22267 22268 22269 22270 22271 22272
## [43] 22273 22274 22275 22276 22277 22278 22279
## [50] 22280 22281 22282 22283
```

¿Que identificadores tienen estas sondas?

```
head(featureNames(gse1397)[control])
```

```
## [1] "AFFX-hum_alu_at"
## [2] "AFFX-HUMGAPDH/M33197_3_at"
## [3] "AFFX-HUMGAPDH/M33197_5_at"
## [4] "AFFX-HUMGAPDH/M33197_M_at"
## [5] "AFFX-HUMISGF3A/M97935_3_at"
## [6] "AFFX-HUMISGF3A/M97935_5_at"
```

Son sondas de control que no corresponden a ningún gen. Hemos dicho antes que queremos modificar los identificadores utilizados en el ExpressionSet. Y queremos hacerlo pasando a Entrez y Ensembl. Buscamos las correspondencias.

```
probeid2entrez =
    AnnotationDbi::select(hgu133a.db,keys=featureNames(gse1397),
                          columns="ENTREZID",keytype="PROBEID")
```

Nos ha devuelvo un `data.frame`.

```
class(probeid2entrez)
```

```
## [1] "data.frame"
```

Es importante notar que no hay una correspondencia 1-1 entre los distintos identificadores.

```
head(probeid2entrez)
```

```
##     PROBEID  ENTREZID
## 1 1007_s_at       780
## 2 1007_s_at 100616237
```

```
## 3   1053_at      5982
## 4    117_at      3310
## 5    121_at      7849
## 6 1255_g_at      2978
```

La primera sonda tiene dos identificadores Entrez distintos. Una opción para resolver esta multiplicidad es utilizar BiocGenerics::match.

```
indices = match(featureNames(gse1397),probeid2entrez$PROBEID)
```

¿Cómo se ha resuelto?

```
head(featureNames(gse1397))

## [1] "1007_s_at" "1053_at"   "117_at"
## [4] "121_at"    "1255_g_at" "1294_at"

head(probeid2entrez$PROBEID,n=7)

## [1] "1007_s_at" "1007_s_at" "1053_at"
## [4] "117_at"    "121_at"    "1255_g_at"
## [7] "1294_at"

head(indices)

## [1] 1 3 4 5 6 7
```

Elige la primera correspondencia e ignora las demas como hemos visto.

```
eset = gse1397
fData(eset) = probeid2entrez[indices,]
```

Podemos comprobar, con Base::all.equal, si son iguales los nombres del ExpressionSet con la columna de identificadores Affymetrix.

```
all.equal(fData(eset)$PROBEID,featureNames(eset))

## [1] TRUE
```

3.10 Ejercicios

* **Ej. 12** — Determinar los códigos Entrez, Gene Ontology y Ensembl para los genes BRCA1 y BRCA2 implicados en el cáncer de mama.

* **Ej. 13** — Se pide encontrar el gen BRCA1 en el ratón (Mus musculus). Utilizad el paquete de anotación [29, org.Mm.eg.db].

Parte II

Datos

Capítulo 4

Microarrays

4.1 Introducción

En este tema tratamos sobre datos de expresión obtenidos utilizando microarrays. Nos ocupamos del procesado (o mejor, del preprocesado) de los datos desde los datos originales (o datos a nivel de sonda) hasta los datos tal y como los hemos estado analizando.

Los datos multtest::golub que hemos usado ya han sido preprocesados. En este tema nos ocupamos de este punto previo y no menor. Muchos de los análisis de expresión diferencial posterior dependen de un modo esencial de lo que hacemos antes. Con frecuencia el experimentador confía en el preprocesado que el fabricante del chip realiza. No necesariamente este preprocesado tiene porqué estar mal hecho pero en cualquier caso es preciso conocerlo y evaluarlo. En lo que sigue veremos como hay funciones que reproducen lo que el fabricante hace.

En este capítulo la referencia de mayor interés es [53]. En particular tienen un interés especial los tres primeros capítulos.

La expresión de un gen es el proceso mediante el que se transcribe el DNA en una serie de copias de mRNA. Los microarrays miden la cantidad de mRNA para cada gen. El valor de la expresión de un gen es una medida de luminiscencia relacionada con el mRNA presente. Esto es para un chip de DNA.

4.2 Affymetrix GeneChip

Es conveniente consultar [146, páginas 93 y siguientes]. Vamos a leer datos obtenidos mediante un escáner Affymetrix GeneChip. Estos arrays de oligonucleótidos tienen un pequeño número de sondas para un mismo gen. Hablaremos de conjunto de sondas refiriéndonos a todas las sondas que en un mismo chip están asociadas con un mismo gen. Cada sonda son pequeñas celdas con oligonucleótidos específicos de un gen. Estos oligonucleótidos tienen 25 pares de bases. Cada oligonucleótido es una parte (aproximadamente) específica de un gen. En principio solo el mRNA de este gen lo reconocerá como propio. Sin embargo, también puede ocurrir que el RNA de otros genes encuentre partes comunes con la sonda y se adjunte. Tendríamos una hibridación no específica. Con objeto de considerar el problema Affymetrix modifica la secuencia original. Concretamente el par de bases

en la posición 13. De este modo una hibridación con esta sonda se considera un acoplamiento erróneo.

Las imágenes digitales con la información original son los ficheros DAT. Una vez procesados por el software que proporciona el fabricante obtenemos un resumen por celda que los guarda en los ficheros CEL. Este fichero es el resultado del escaneo y la segmentación de la imagen obtenida. Todavía no hemos resumido estas intensidades de los píxeles en un valor por gen. En estos ficheros tenemos la media y desviación estándar de los niveles de gris así como la localización de la sonda dentro del array. Se necesita también conocer la correspondencia entre sondas y nombres de los genes. Esta información aparece en el fichero CDF.[1]

La información que se maneja se puede considerar a tres niveles:

Imagen El primer nivel es de la imagen digital (ficheros DAT en Affymetrix GeneChip). Hay que utilizar técnicas de procesado de imagen digital para obtener a partir de esta imagen otra imagen donde cada pixel tiene como nivel de gris la expresión de la sonda en esa celda.

Datos a nivel de sonda Tenemos un imagen digital donde un pixel contiene una cuantificación de la cantidad de hibridación.

Datos de expresión Ya tenemos corregido el fondo y normalizadas las muestras. Tenemos una matriz de expresión de modo que en la fila tenemos el gen y en la columna la muestra. A partir de aquí se trabaja para analizar la posible actividad diferenciada de un gen o un conjunto de genes.

4.3 Obteniendo los datos

En esta sección vamos a utilizar los datos con número de acceso GEO GSE21779. [77]

[77] Ver § 5.9.

4.4 Sonda y conjunto de sondas

En estos microarrays tenemos distintas sondas correspondientes al mismo gen. ¿Qué chip o plataforma se ha utilizado se ha utilizado?

Podemos verlo con Biobase::annotation.

```
pacman::p_load(Biobase,affy)
annotation(gse21779raw)

## [1] "hgu133plus2"
```

El número de sondas y de muestras lo podemos conocer con

```
dim(exprs(gse21779raw))

## [1] 1354896      18
```

[1] Cuando leamos datos veremos que si no tenemos cargado el fichero CDF necesario el programa los baja sin que se lo pidamos.

Nos fijamos en el primer array cuyas expresiones aparecen en la primera columna. Podemos ver las intensidades del primer microarray en forma de una imagen a niveles de gris. Cuando más blanco es el punto mayor es la intensidad observada en la sonda y por lo tanto mayor la expresión del gen al que está asociada. Cada pixel corresponde con una sonda distinta. La imagen la tenemos en la figura 4.1.

```
png(paste0(dirTamiFigures,"microarray4.png"))
image(gse21779raw[,1])
dev.off()

## pdf
##   2
```

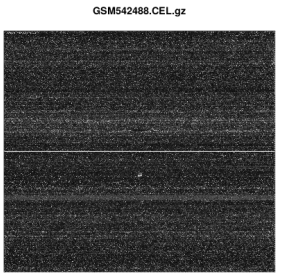


Figura 4.1: Imagen a niveles de gris correspondiente al primer array de los datos gse21779.

Podemos conocer las identificadores que Affymetrix le ha asignado a estas sondas con

```
probeNames(gse21779raw)
```

Nos fijamos en la sonda que aparece en la posición 400.

```
ID = probeNames(gse21779raw)[400]
```

El número de sondas que componen el conjunto asociado a este identificador es

```
sum(probeNames(gse21779raw) == ID)

## [1] 11
```

Podemos ver los cardinales de todos los conjuntos de sondas que lleva el chip.

```
counts = table(probeNames(gse21779raw))
table(counts)

## counts
##     8     9    10    11    13    14    15    16
##     5     1     6 54130     4     4     2   482
##    20    69
##    40     1
```

4.5 Variabilidad intra y entre arrays

Dentro de todo el vector con los nombres de las sondas buscamos las que ocupa la primera tanto para PM como para MM (de ahí el argumento `both`).

```
indexProbes(gse21779raw,"both",ID)[[1]]

##  [1] 1088751  883561 1054203  366753  178855
##  [6]   68793  490689   62456 1123802  939573
## [11] 1349651 1089915  884725 1055367  367917
## [16]  180019   69957  491853   63620 1124966
## [21]  940737 1350815
```

¿Cuáles corresponden a PM?

```
(posiciones = indexProbes(gse21779raw,"pm",ID)[[1]])

##  [1] 1088751  883561 1054203  366753  178855
##  [6]   68793  490689   62456 1123802  939573
## [11] 1349651
```

Tenemos 11 correspondientes a PM. Y las demás corresponden a MM:

```
indexProbes(gse21779raw,"mm",ID)[[1]]

##  [1] 1089915  884725 1055367  367917  180019
##  [6]   69957  491853   63620 1124966  940737
## [11] 1350815
```

En figura 4.2(a) vemos dónde está localizado el conjunto de sondas[78] dentro de la imagen. En cada localización tenemos un valor de PM y otro valor MM.

[78] Probe set.

Vamos a representar los niveles de expresión PM correspondientes a un mismo conjunto de sondas de un array dado (figura 4.2(b)).

```
library(ggplot2)
pmID = probes(gse21779raw[,1],"pm",ID)
df1 = data.frame(sondas = 1:11,intensidad = pmID[,"GSM542488.CEL.gz"])
ggplot(df1,aes(x=sondas,y=intensidad))+geom_line()
```

En abscisas tenemos el número de la sonda (de un mismo conjunto) y en ordenadas su expresión. Si no hubiera ningún tipo de ruido lo que se debe observar sería una línea horizontal, unos mismos niveles de expresión. Pero no es así.

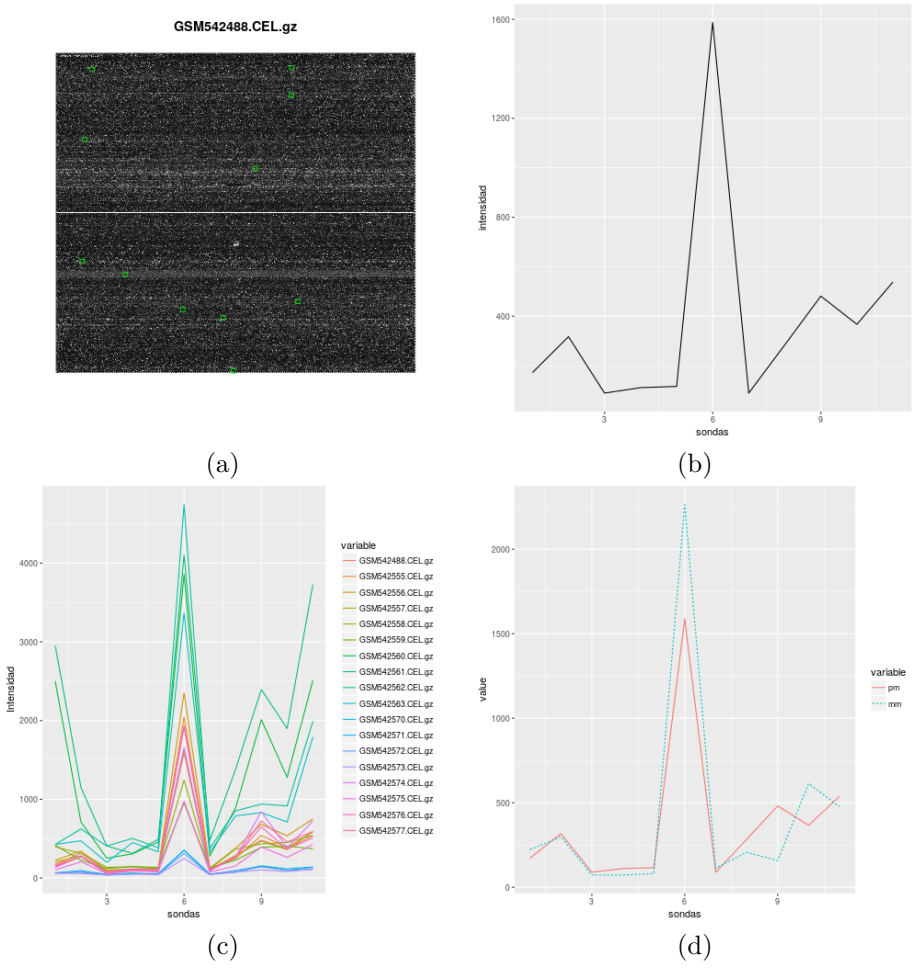
El dibujo 4.2(b) se refiere a la variabilidad **intra array** (utilizando las sondas de un mismo conjunto) para un mismo gen. Vamos superponer (una línea por muestra) las expresiones del mismo gen pero para todas las muestras. De este modo ilustramos la variabilidad entre muestras. En la figura 4.2(c) mostramos este dibujo.[79]

[79] Observad el uso de la función reshape::melt para obtener el data.frame.

```
pm0 = probes(gse21779raw,"pm",ID)
pm0 = data.frame(pm0,sondas=1:11)
pm1 = reshape::melt(pm0,id="sondas")
ggplot2::ggplot(data=pm1,aes(x=sondas,y=value,colour=variable))+
    geom_line()+ylab("Intensidad")
```

```
png(paste(dirTamiFigures,"gse217791arrays.png",sep=""))
pm0 = probes(gse21779raw,"pm",ID)
pm0 = data.frame(pm0,sondas=1:11)
pm1 = reshape::melt(pm0,id="sondas")
ggplot2::ggplot(data=pm1,aes(x=sondas,y=value,colour=variable))+
    geom_line()+ylab("Intensidad")
dev.off()
```

En la figura 4.2(c) cada línea corresponde a un array y los puntos que representamos en cada línea es un conjunto de sondas. Este conjunto de sondas es el mismo que elegimos en todos los arrays.



(a) (b) (c) (d)

Figura 4.2: (a)Image CEL correspondiente al primer array o chip. Cada pixel corresponde con una sonda distinta. Los cuadrados verdes nos localizan un conjunto de sondas asociadas al mismo gen. (b) Los valores (o intensidades) PM para un grupo de sondas en el primer array de nuestras muestras. (c) Los valores (o intensidades) PM para un mismo grupo de sondas en los distintos arrays. Cada línea corresponde con un array distinto. En abscisas tenemos la misma sonda en cada uno de los arrays. (d) En la primera muestra de gse21779 mostramos los valores PM y, en las mismas localizaciones, los valores MM. No siempre PM es mayor que MM como sería de esperar.

¿Qué significa este dibujo? ¿Cómo lo podemos interpretar? Si las distintas sondas asociadas al mismo gen nos dieran una misma expresión cada una de las líneas debía ser horizontal. No es así como hemos visto para la primera y para todas las demás. Tenemos ahora una segunda fuente de variación. Estas muestras se han tomado bajo tres condiciones distintas. Si no hubiera diferencia entre condiciones las líneas debieran de estar muy superpuestas.

De acuerdo al diseño de este chip cabe esperar que en una misma localizacion el valor PM sea mayor que el valor de MM. Esto es lo esperable ya que PM es señal, estamos observando la hibridación específica (aquella para la que la sonda ha sido diseñada). El valor de MM lo que mide es ruido óptico e hibridación no específica, en resumen, ruido en un sentido amplio. La señal debe dominar al ruido, el valor de PM debiera de ser mayor que el valor de MM en cada celda. Esto sobre el papel. Pero no en la realidad. Veamos (figura 4.2(d)) para un mismo array y para un conjunto de genes dados los valores de PM y los valores MM.

```
library(reshape)
pmm = data.frame(sondas=1:11,pm = probes(gse21779raw[,1],"pm",ID),
    mm = probes(gse21779raw[,1],"mm",ID))
df2 = reshape::melt(pmm,id="sondas")
levels(df2[,"variable"]) = c("pm","mm")
ggplot(df2,aes(x=sondas,y=value,colour=variable,linetype=variable))
+geom_line()
```

Podemos hacer una valoración global para todos los arrays que tenemos. ¿Qué proporción de sondas son tales que el valor PM es menor que el correspondiente valor MM?

```
table(probes(gse21779raw,"pm") >= probes(gse21779raw,"mm"))

##
##   FALSE    TRUE
## 4460757 6415887
```

Vemos que tenemos un 41 % en este caso particular. No es poco.

4.6 Control de calidad

Vamos a estudiar distintos procedimientos que intentan evaluar los distintos microarrays que componen nuestro experimento. El objetivo es decidir si es necesario descartar total o parcialmente alguna de las muestras.

4.6.1 Dibujo de la media-diferencia de Tukey o dibujo de Bland-Altman

El dibujo media-diferencia de Tukey es más habitualmente conocido como el dibujo Bland-Altman estos autores lo popularizaron utilizándolo en una revista médida.[80]

Supongamos que tenemos pares de medidas (x_i, y_i) con $i = 1, \ldots, n$ y pretendemos ver el grado de coincidencia de las mismas. O, dicho de otro modo, si valores mayores corresponden con diferencias mayores

[80] El trabajo completo (con correcciones respecto de la publicación original) es Bland-Altman y corresponde con la referencia [15].

entre las cantidades observadas. Es de suponer que dos procedimientos que coincidan tendrán un coeficiente de correlación alto. Sin embargo, este valor viene condicionado por la variabilidad de las medidas. El dibujo media-diferencia de Tukey o dibujo de Bland-Altman es una representación gráfica que sirve para valorar el grado de coincidencia de los pares de medidas. Supongamos que tenemos los pares (x_i, y_i) con $i = 1, \ldots, n$ entonces consideramos los puntos que tienen coordenadas

$$\left(\frac{x_i + y_i}{2}, (x_i - y_i)\right).$$

Es decir, en abscisas tenemos la media de las dos medidas $\frac{x_i+y_i}{2}$ y en ordenadas tenemos su diferencia, $x_i - y_i$.

En ocasiones, este dibujo se completa con dos líneas horizontales. Si $d_i = x_i - y_i$ entonces podemos completar la representación gráfica añadiendo dos líneas horizontales correspondientes con los límites del intervalo de confianza para la diferencia media entre los dos procedimientos esto es las líneas tendrían como ordenadas $\bar{d} \pm t_{n-1,1-\frac{\alpha}{2}} \frac{S}{\sqrt{n}}$ donde S es la desviación estándar de las diferencias d_i y $t_{n-1,1-\frac{\alpha}{2}}$ es el percentil $1 - \frac{\alpha}{2}$ de una t de Student con $n - 1$ grados de libertad.

4.6.2 MA plots

Estos dibujos sirven para determinar si alguno de los microarrays debe de ser descartado del estudio y si necesitamos homogeneizar los valores para las distintas muestras. Son una aplicación del dibujo media-diferencia propuesto por Tukey.[81]

[81] Es conveniente consultar http://en.wikipedia.org/wiki/MA_plot.

Trabajamos, en lugar de con las intensidades originales observadas, con su logaritmo en base 2. Supongamos que u_i y v_i denotan las intensidades originales en la sonda i-ésima. Consideramos $x_i = \log_2 u_i$ e $y_i = \log_2 v_i$ y estos son los valores a comparar. En este contexto se habla de dibujos MA ya que

$$m_i = x_i - y_i = \log_2 \frac{u_i}{v_i},$$

y

$$a_i = \frac{1}{2}(x_i + y_i) = \frac{1}{2} \log_2 u_i v_i = \log_2 \sqrt{u_i v_i}.$$

Una primera opción es comparar todos los pares de microarrays utilizando el dibujo media-diferencia de Tukey.

Una segunda opción consiste en considerar una especie de microarray *típico* y comparar los demás con este. Por ejemplo, una opción que implementa el paquete [70, affy] en la función affy::MAplot consiste en calcular para cada sonda la mediana a lo largo de todos los microarrays que componen nuestro experimento. Por tanto, la mediana de las intensidades será x_i y el valor y_i corresponde a cada uno de los microarrays de nuestro experimento.

Ejemplo 4.1 (MA plot) *Vamos a ilustrar el uso de los dibujo MA con los datos* **gse21779**. *En la figura 4.3 podemos ver una comparación entre los dos primeros microarrays del experimento.*

```
MAplot(gse21779raw[,1:2],pairs=TRUE,plot.method="smoothScatter")
```

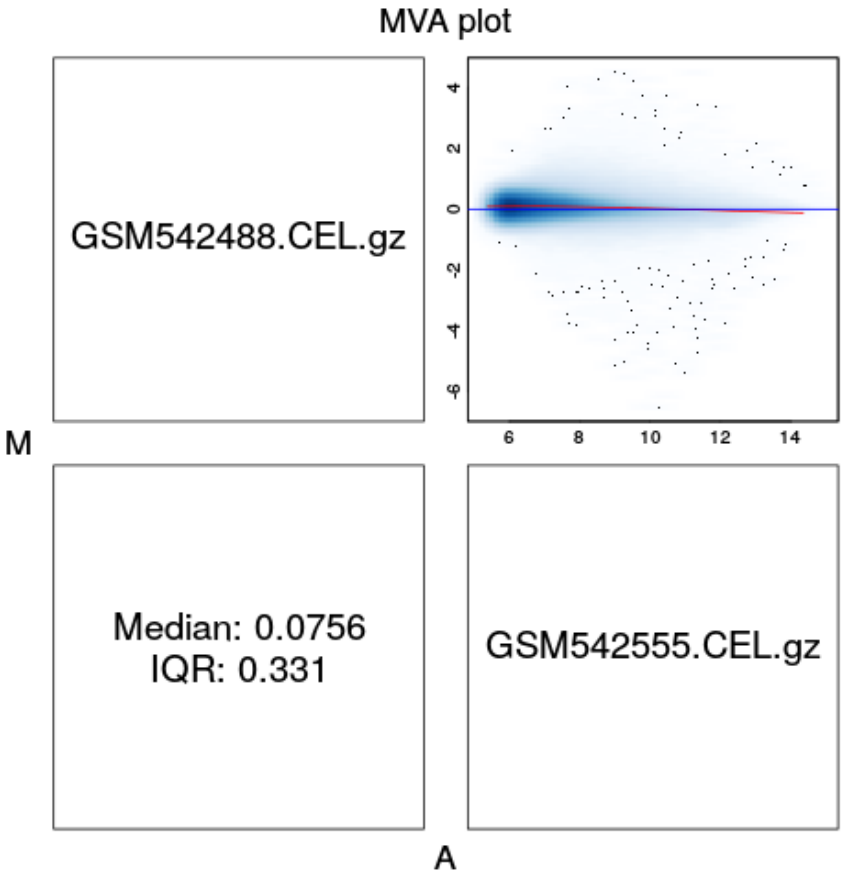
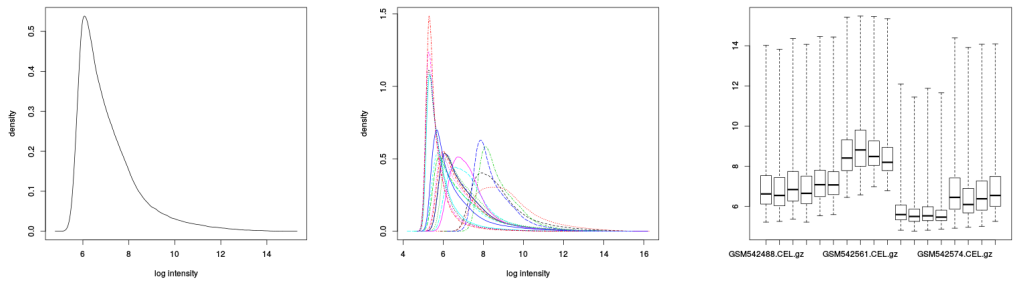


Figura 4.3: MAplot comparando los dos primeros microarrays.

4.6.3 Densidades y diagramas de cajas

En los dos dibujos anteriores hemos ilustrado la variabilidad intra y entre arrays para un mismo gen. Tomemos un punto de vista más global. En particular, vamos a mostrar el comportamiento de los niveles de expresión para todas las sondas en un array, por ejemplo, el primero de ellos. Estamos evaluando la posibilidad de que alguno de ellos tenga un nivel de ruido inaceptable y debiera de ser eliminado del estudio.



(a) (b) (c)

Figura 4.4: a)Estimador kernel de la densidad de las expresiones a nivel de sonda en el primer microarray. b) Estimadores kernel de la densidad de las intensidades para los distintos microarrays. Vemos que hay diferencias claras. c) Diagramas de cajas de las expresiones a nivel de sonda para los datos gse21779.

Empezamos mostrando en la figura 4.4(a) la densidad de los niveles de expresión en el primer array. Lo hacemos utilizando un estimador kernel de la densidad.

```
affy::hist(gse21779raw[,1])
```

Aunque la función que utilizamos es `hist` (de histograma), no es histograma.[2] Es un estimador kernel de la densidad. Con objeto de ver (y comparar) la variabilidad en los distintos arrays podemos estimar los estimadores kernel de las densidades de las expresiones en cada uno de los arrays que componen nuestra muestra. Lo tenemos en la figura 4.4(b).

```
affy::hist(gse21779raw)
```

Llama la atención la gran diferencia entre estos estimadores de la densidad para distintas muestras. En cualquier caso en todos ellos vemos una clara asimetría a la derecha. Obviamente corresponde a los genes con un alto nivel de expresión.

De un modo análogo podemos para evaluar la variabilidad las expresiones en cada arrays y entre arrays podemos utilizar diagramas de cajas. Se muestra en la figura 4.4(c).

```
affy::boxplot(gse21779raw)
```

En este caso los genes con una alta (anormalmente grande) expresión corresponden con el bigote superior en los diagramas de caja que acabamos de representar.

[2] Una mala elección de nombre.

4.6.4 Utilizando arrayQualityMetrics

El paquete [75, arrayQualityMetrics] nos permite un control de calidad sencillo utilizando arrayQualityMetrics::arrayQualityMetrics. [82]

[82] El informe correspondiente lo tenemos en http://wwww.uv.es/ayala/docencia/tami/reports/Report_for_gse21779/index.html.

```
library(arrayQualityMetrics)
arrayQualityMetrics(expressionset = gse21779raw,
outdir = "Report_for_gse21779raw", ## Fija directorio de salida
force = TRUE,  ## Sobreescribe
do.logtransform = TRUE) ## Transforma a log2
```

4.6.5 Corrigiendo errores técnicos

Si algo queda claro en esta sección es la necesidad de tratar los datos antes de poder compararlos y obtener alguna conclusión biológica. Es decir, hemos de trabajar los valores observados a nivel de sonda antes de obtener un resumen que es la expresión del gen.

De hecho podemos ver los mínimos de PM en las distintas muestras.

```
apply(probes(gse21779raw,"pm"),2,min)
```

```
## GSM542488.CEL.gz GSM542555.CEL.gz
##               32               31
## GSM542556.CEL.gz GSM542557.CEL.gz
##               34               33
## GSM542558.CEL.gz GSM542559.CEL.gz
##               21               39
## GSM542560.CEL.gz GSM542561.CEL.gz
##               67               60
## GSM542562.CEL.gz GSM542563.CEL.gz
##               74               82
## GSM542570.CEL.gz GSM542571.CEL.gz
##               26               27
## GSM542572.CEL.gz GSM542573.CEL.gz
##               26               27
## GSM542574.CEL.gz GSM542575.CEL.gz
##               28               29
## GSM542576.CEL.gz GSM542577.CEL.gz
##               28               32
```

Y vemos como efectivamente no son nulos. La *corrección de fondo* pretende que aproximadamente sean nulos los valores más pequeños. Esto correspondería con aquellos genes que no tienen actividad.

También hemos visto que la distribución de las intensidades es claramente muy diferente entre arrays. Lo mostraban los estimadores kernel y los diagramas de cajas. El proceso de hacer comparables los datos entre arrays es lo que se conoce como *normalización*.

Una vez hemos corregido fondo y normalizadas las muestras, seguimos teniendo para cada gen distintas sondas en cada array. Toca resumir estos valores en un solo valor que indique la expresión del gen asociado al conjunto de sondas.

Estas tres operaciones pueden realizarse de un modo secuencial o bien hay métodos que las realizan de un modo combinado. Hay

publicados muchos métodos. Lo que vamos a utilizar en este capítulo está en el paquete [70, affy].[83]

En estas notas vamos a estudiar con detalle dos métodos. El primero es un método que corrige cada chip, es el método MAS5.0 que es el que proporciona el software de Affymetrix. Muchos trabajan con los datos que directamente les da este software y por tanto es interesante conocerlo. Lo tratamos en § 4.7. El segundo es el método RMA. Este procedimiento trabaja simultáneamente con todos los chips del experimento. Es pues un procedimiento multiarray o multichip. Este método lo estudiamos en § 4.8.

El objetivo ahora es considerar procedimientos que corrigen las variabilidades entre los microarrays intentando conseguir un valor de expresión asociado a cada gen sin esos ruidos de caracter técnico.

[83] Es especialmente interesante consultar la viñeta builtinMethods.pdf que aparece en la ayuda del paquete *affy*. Como material complementario es de interés [53, capítulo 2].

4.7 Método MAS5

Veremos cómo pasar de los datos a nivel de sonda a los valores de expresión del gen. Mucho de lo que luego analizamos depende de cómo se realiza este paso. Lo que se comenta es específico para Affymetrix GeneChip. En cada localización tendremos sondas que son oligonucleótidos específicos de un gen. De hecho son pares de sonda: una con la secuencia deseada y otra en la que se modifica la base en la posicion 13. La intensidad (o expresión) observada con el oligonucleótido correcto le llamamos PM (perfect matching) y el observado sobre el modificado le llamamos MM (miss matching). Se pensaba que si a PM le restábamos MM corregiríamos el ruido de fondo. Esto no es cierto, una cierta proporción de localizaciones muestran un valor MM mayor que el valor PM, lo que en principio va en contra de lo que lógicamente debería producirse.

Los datos y la información de un Affymetrix GeneChip se almacenan en distintos ficheros. En el fichero CEL tenemos la intensidad media de los pixels de la sonda, la desviación estándar e información de dónde se sitúa la sonda dentro del array. Cada array lleva asociado un fichero CEL. Para poder interpretar el significado biológico de las sondas se necesita un mapa indicando a qué gen corresponde cada gen. Esta información está en el fichero CDF. Este fichero depende del chip utilizado y no varía para distintas muestras.

En esta sección explicamos el método MAS5.0 y nos basamos en el manual http://media.affymetrix.com/support/technical/whitepapers/sadd_whitepaper.pdf.[3]

Empezamos a trabajar a partir del fichero .CEL que ha sido generado por el software Microarray Suite.

4.7.1 Corrección de fondo

Se pretende determinar unos valores para restarlos a las intensidades originales en cada celda.

Determinan valores por zonas

1. Se divide el array en K zonas rectangulares: Z_k con $k = 1, \ldots, K$.
2. En cada Z_k se ordenan las intensidades observadas en las distintas celdas y se considera el 2% con intensidades menores. Se toma como medida de fondo la media muestral

[3] Como casi todos los manuales técnicos no es prodigio de claridad precisamente.

de las intensidades previamente consideradas. Denotamos esta valor como $b(Z_k)$.

3. Del mismo modo que en punto anterior, se considera en cada Z_k ese 2% de celdas con intensidades menores y calcular su desviación estándar que denotamos $s(Z_k)$.

Suavizando el ajuste Sea la celda localizada en (x, y) y vamos a denotar por $d_k(x, y)$ la distancia desde (x, y) al centro de la región Z_k. Consideramos la siguiente cantidad

$$w_k(x, y) = \frac{1}{d_k^2(x, y) + s_0}$$

donde s_0 nos evita que el cociente se pueda anular y por defecto toma el valor $s_0 = 100$. Para la celda localizada en (x, y) consideramos

$$b(x, y) = \frac{1}{\sum_{k=1}^{K} w_k(x, y)} \sum_{k=1}^{K} w_k(x, y) b(Z_k)$$

Corrección del ruido Ahora vamos a bajar las intensidades restándole una medida de ruido de fondo local. El problema que se puede producir es que nos encontremos con valores negativos. Este valor de ruido en la celda localizada en (x, y) se calcula como

$$n(x, y) = \frac{1}{\sum_{k=1}^{K} w_k(x, y)} \sum_{k=1}^{K} w_k(x, y) s(Z_k).$$

La idea ahora es quitar a las intensidades originales el fondo $(b(x, y))$ que hemos calculado pero asegurándonos que no nos surja valores negativos. La intensidad original se denota como $I(x, y)$ en la localización (x, y). El valor ajustado de la intensidad viene dado por

$$A(x, y) = \max\{I(x, y) - b(x, y), f_0 n(x, y)\}$$

El parámetro f_0 se suele fijar en $f_0 = 0.5$.[4]

4.7.2 Cálculo del valor de expresión

Por el procemiento descrito anteriormente podemos ajustar los valores de PM y de MM. En lo que sigue se supone que hemos ajustado estos valores.

Lo esperable y correcto es que en una celda tengamos un valor de PM mayor que el correspondiente valor MM. Esto no es así en muchas ocasiones como hemos visto. Se define un valor IM (Ideal Mismatch)[5] que intenta corregir este problema.

Denotamos por (PM_{ij}, MM_{ij}) el valor de PM y de MM de la j-ésima sonda perteneciente al i-ésimo conjunto de sondas (formado por todas las sondas asociadas a un mismo gen).

[4] En el manual hay una igualdad incomprensible que dice $I(x, y) \max\{I(x, y), 0.5\}$. En fin, ya veremos qué es exactamente lo que quieren hacer. He puesto lo que sensatamente parece que quieren hacer.

[5] Un apaño se decía antes.

Se calcula un valor que llaman background específico para cada conjunto de sondas.

$$\boldsymbol{B}_i = T_{bi}\{\log_2(PM_{ij}) - \log_2(MM_{ij}) : j = 1, \ldots, n_i\},$$

donde T_{bi} denota el algoritmo bipesado en un solo paso (§ 21.1). Básicamente un promedio robusto de las diferencias indicadas que no son más que logaritmos de sus cocientes. ¿Qué estima este valor $\boldsymbol{B}_i$ para el conjunto de sondas i? Si el valor de

$$IM_{ij} = \begin{cases} MM_{ij} & \text{si } MM_{ij} < PM_{ij} \\ \frac{PM_{ij}}{2^{\boldsymbol{B}_i}} & \text{si } MM_{ij} \geq PM_{ij} \text{ y } \boldsymbol{B}_i > \alpha \\ \frac{PM_{ij}}{2^{\frac{\alpha}{1+(\alpha-\boldsymbol{B}_i)/\beta}}} & \text{si } MM_{ij} \geq PM_{ij} \text{ y } \boldsymbol{B}_i \leq \alpha \end{cases}$$

Por defecto se toma $\alpha = 0.03$ y $\beta = 10$.

Ahora calculamos

$$V_{ij} = \max\{PM_{ij} - IM_{ij}, d\}$$

siendo el valor por defecto para d, $d = 2^{-20}$.

Transformamos este valor con

$$\mathbb{V}_{ij} = \log_2(V_{ij})$$

con $j = 1, \ldots, n_i$. Y volvemos a aplicar un estimador biponderado en un solo paso (§ 21.1).

$$SignalLogValue_i = T_{bi}(\mathbb{V}_{i1}, \ldots, \mathbb{V}_{in_i}).$$

Fijamos una señal objetivo, Sc, que por defecto se toma $Sc = 500$. Se calcula el siguiente factor de escala para el grupo de sondas i.

$$sf_i = \frac{Sc}{MediaAjustada\{2^{SignalLogValue_i}, 0.02, 0.98\}}$$

En la anterior expresión MediaAjustada indica la media ajustada (media de los valores quitando una cierta proporción de los más grandes y de los más pequeños) en donde se quita el 2% de los más grandes y el 2% de los más pequeños.

El valor final para el conjunto de sondas i es

$$ReportedValue(i) = sf_i \times 2^{SignalLogValue_i}$$

Se incorpora una posibilidad de normalización que no indicamos.

El método **plier** no lo tratamos en estas notas. Está implementado en el paquete [**R-plier**].

4.8 Robust multichip average (RMA)

Una referencia con muchos más detalles de los que vemos en esta sección es [18, pág. 41-59] En este método trabajamos con todos los arrays simultáneamente[84] a diferencia del método anterior donde se procesaba cada array independientemente. El procedimiento solamente utiliza los valores PM y no utiliza para nada los valores MM.

[84] Y de ahí su nombre **R**obust **M**ultiarray **A**verage.

4.8.1 Corrección de fondo

Se empieza calculando una corrección de fondo.[85] Asumimos (sugerido por la distribución empírica observada en las intensidades de las sondas) que los valores PM pueden considerarse como la suma de una variable Y con distribución normal con media μ y varianza σ^2 (que modeliza el ruido) y de una variable X con distribución exponencial con media α (que modelizaría la señal). Por tanto el valor observado aleatorio S sería

$$S = X + Y$$

con $X \sim Exp(\alpha)$ e $Y \sim N(\mu, \sigma^2)$. La normal se trunca en cero para evitar valores negativos. Si S denota la intensidad aleatoria en la sonda entonces, para un valor observado $S = s$, se tiene

$$E(X|S = s) = a + b\frac{\phi(\frac{a}{b})}{\Phi(\frac{a}{b})}$$

siendo $a = s - \mu - \sigma^2\alpha$ y $b = \sigma$ mientras que ϕ y Φ denotan las funciones de densidad y de distribución de una normal estándar (con media 0 y varianza 1). Los parámetros del modelo (α, μ, σ^2) se estiman mediante un procedimiento no paramétrico. Utilizando las intensidades observadas en las sondas PM se estima la moda de la distribución. Aquellos valores por encima de la moda son utilizados para estimar α y los puntos por debajo de la moda se utilizan para estimar los parámetros μ y σ^2.

[85] Que esencialmente no se ha publicado. Aparece una descripción más o menos vaga en [68].

4.8.2 Normalización de cuantiles

En este paso se pretende conseguir que las distribuciones empíricas de las expresiones a nivel de sonda de los distintos arrays sean la misma.[6] [86] Esencialmente lo que buscamos es que las expresiones que se observan en los distintos microarrays sean las mismas y con las mismas frecuencias.

La normalización que se aplica en este método es una *normalización de cuantiles*.[87]

Veamos qué es con un ejemplo detallado y sencillo.

1. Nuestro experimento tiene dos conjuntos de sondas y cada una con tres sondas. Además suponemos que tenemos simplemente tres chips. En la siguiente tabla recogemos las expresiones.

Conjunto de sondas	Sonda	Chip 1	Chip 2	Chip 3
1	1	7	9	19
1	2	3	5	14
1	3	2	6	11
2	1	4	8	8
2	2	10	11	16
2	3	12	10	15

2. Determinamos los valores mayores en cada chip y calculamos el promedio.

[86] En § 1.9 se recuerda el concepto de distribución empírica.

[87] Quantile normalization [18].

[6] Recordemos que en este método solamente utilizamos las expresiones de las sondas PM.

```
(12 + 11 + 19)/ 3

## [1] 14
```

Sustituimos los valores originales por estos promedios.

Conjunto de sondas	Sonda	Chip 1	Chip 2	Chip 3
1	1	7	9	**14**
1	2	3	5	14
1	3	2	6	11
2	1	4	8	8
2	2	10	**14**	16
2	3	**14**	10	15

3. Consideramos los segundos valores mayores en cada chip y los promediamos.

```
(10 + 10 + 16) /3

## [1] 12
```

Sustituimos los valores originales por los promedios.

Conjunto de sondas	Sonda	Chip 1	Chip 2	Chip 3
1	1	7	9	14
1	2	3	5	14
1	3	2	6	11
2	1	4	8	8
2	2	**12**	14	**12**
2	3	14	**12**	15

4. Consideramos los terceros valores mayores en cada chip.

```
(7 + 9 + 15)/3

## [1] 10.33333
```

Sustituimos los valores originales por los promedios.

Conjunto de sondas	Sonda	Chip 1	Chip 2	Chip 3
1	1	**10.33**	**10.33**	14
1	2	3	5	14
1	3	2	6	11
2	1	4	8	8
2	2	12	14	12
2	3	14	12	**10.33**

5. Y así sucesivamente hasta llegar al sexto valor que ya es el más pequeño en cada chip. Los datos finales son los de la siguiente tabla.

Conjunto de sondas	Sonda	Chip 1	Chip 2	Chip 3
1	1	10.33	10.33	14
1	2	6.66	5	8.66
1	3	5	6.66	6.66
2	1	8.66	8.66	5
2	2	12	14	12
2	3	14	12	10.33

Una vez ilustrado el método es sencillo de explicar.

1. Tomamos n vectores de longitud N y construimos la matriz X $N \times n$ que los tiene por vectores columna.
2. Ordenamos cada columna de X separadamente y tenemos X_s.
3. Calculamos la media por filas de la matrix X_s y creamos X'_s una matriz con la misma dimensión que X, tal que en la fila j tenemos la media de la fila de X_s repetida n veces.
4. Obtenemos X_t en donde cada columna de X'_s recupera el orden original.

¿Qué estamos haciendo desde el punto de vista probabilístico? Hemos considerado los cuantiles muestrales en cada columna. Hemos considerado la media muestral de estos cuantiles muestrales. Supongamos que F es la función de distribución de estas medias de cuantiles muestrales. Tomamos G la función de distribución muestral o empírica de las expresiones de cada uno de los microarrays. Si la expresión original la denotamos por y_i para la i-ésima sonda entonces la nueva expresión de dicha sonda viene dada por $y'_i = F^{-1}(G(y_i))$. Cuando F corresponde a la uniforme en el intervalo $[0, 1]$ esto recibe el nombre de transformación integral de la probabilidad.

4.8.3 Resumen

Ya hemos realizado la corrección de fondo (§ 4.8.1). A los valores obtenidos les hemos aplicado una normalización de cuantiles (§ 4.8.2). Ahora nos queda obtener el resumen de las distintas sondas dentro de cada conjunto (de sondas) y para cada microarray. Se aplica el método median polish de Tukey.[88] Previamente necesitamos algo de notación. En cada array tendremos N sondas y suponemos que tenemos n arrays. Por tanto la matriz de expresión a nivel de sonda será: $\boldsymbol{x} = [x_{ij}]_{i=1,\ldots,N;j=1,\ldots,n}$. La sonda i-ésima en el array j-ésimo tiene una expresión o intensidad x_{ij}. Las sondas las suponemos agrupadas en grupos correspondientes a un mismo gen. Denotamos el grupo k-ésimo de sondas con S_k. Por tanto, $S_k \subset \{1, \ldots, N\}$. Por ejemplo, el conjunto $S_k = \{i_1, \ldots, i_{|S_k|}\}$, es decir, corresponde con las filas $\{i_1, \ldots, i_{|S_k|}\}$ de la matrix $\boldsymbol{x}$ original. Por simplificar la notación tomaremos $y_{rj} = x_{i_r,j}$ y por tanto

[88] § 21.2.

$$\boldsymbol{y} = [y_{ij}]_{i=1,\ldots,|S_k|,j=1,\ldots,n} = [x_{ij}]_{i\in S_k,j=1,\ldots,n}.$$

En resumen, la matriz $\boldsymbol{y}$ simplemente corresponde con las filas de la matrix $\boldsymbol{x}$ correspondiente al grupo de sondas S_k y todas las columnas.

En lo que sigue nos referimos al grupo k-ésimo de sondas y lo que se asume es para cada grupo. Asumimos el siguiente modelo:

$$\log_2(y_{ij}) = \mu + \theta_j + \alpha_i + \epsilon_{ij} \tag{4.1}$$

con las siguientes restricciones

1. la mediana de $\{\theta_j : j = 1, \ldots, n\}$ es nula,
2. la mediana de $\{\alpha_i : i = 1, \ldots, |S_k|\}$ es nula
3. la mediana de $\{\epsilon_{ij} : j = 1, \ldots, n\}$ sea nula para cada i,

4. la mediana de $\{\epsilon_{ij} : i = 1, \ldots, |S_k|\}$ sea nula para cada j.

El método median polish es un procedimiento para estimar los parámetros anteriores que aparecen en las ecuaciones 4.1. ¿Cómo? Consideramos, para cada grupo de sondas (por ejemplo, el k-ésimo S_k), la matriz

$$\begin{array}{ccc|c} \delta_{1,1} & \cdots & \delta_{1,n} & a_1 \\ \vdots & \ddots & \vdots & \vdots \\ \delta_{|S_k|,1} & \cdots & \delta_{|S_k|,n} & a_{|S_k|} \\ \hline b_1 & \cdots & b_n & m \end{array}$$

En esta matriz las sondas corresponden a las filas y las columnas a los arrays. Además añadimos una columna y una fila (las últimas separadas por una línea continua) en la que aparecen los efectos de fila y columna. Aplicamos el siguiente procedimiento iterativo.

1. Fijamos $\delta_{ij} = \log_2(y_{ij})$, $a_i = b_i = 0\ \forall i, j$.
2. Calculamos la mediana para cada fila a lo largo de las columnas ignorando la última columna (separada por la línea).
3. Restamos a cada elemento de la fila la mediana correspondiente. Esta mediana se suma a la última columna (formada por $a_1, \ldots, a_{|S_k|}, m$).
4. Calculamos la mediana para cada columna de los valores correspondientes a las distintas filas sin considerar la última fila.
5. Restamos la mediana calculada en el paso anterior a cada elemento de la columna exceptuando la última fila. A la última fila sumamos las medianas calculadas.
6. El proceso descrito en los cuatro pasos anteriores continua iterando sucesivamente entre filas y columnas hasta que los cambios que se producen son nulos o muy pequeños.
7. Una vez finalizado el proceso iterativo tendremos $\hat{\mu} = m$, $\hat{\theta}_j = b_j$ y $\hat{\alpha}_i = a_i$. El valor δ_{ij} será el estimador de ϵ_{ij}: $\hat{\epsilon}_{nij} = \delta_{ij}$.

Una vez estimados estos parámetros los estimadores (en escala logarítmica en base 2) de la expresión del grupo de sondas k en el array j viene dada por

$$\hat{\mu} + \hat{\theta}_j.$$

Es un método que no nos proporciona estimaciones de error de los estimadores. Es muy rápido de implementar. Al utilizar medianas es menos sensibles a observaciones extremas. Sin embargo, el resultado puede ser distinto según empecemos por filas o columnas.

4.9 MAS5 y RMA con affy

En el paquete [70, affy] tenemos implementados los dos procedimientos comentados en las dos secciones previas. Usamos las función affy::rma y affy::mas5.

```
gse21779_rma = affy::rma(gse21779raw)
gse21779_mas5 = affy::mas5(gse21779raw)
```

```
save(gse21779_rma,file=paste(dirTamiData,"gse21779_rma.rda",sep=""))
save(gse21779_mas5,file=paste(dirTamiData,"gse21779_mas5.rda",sep=""))
```

```
load(paste(dirTamiData,"gse21779_rma.rda",sep=""))
load(paste(dirTamiData,"gse21779_mas5.rda",sep=""))
```

En la figuras 4.5(a) y 4.5(b) tenemos los estimadores kernel de densidad de los niveles de expresión para los distintos chips una vez hemos aplicado el método RMA y MAS5. En las figuras 4.5(c) y 4.5(d) mostramos los diagramas de cajas correspondientes a los mismos datos preprocesados.

```
library(geneplotter)
geneplotter::multidensity(exprs(gse21779_rma))
geneplotter::multidensity(exprs(gse21779_mas5))
graphics::boxplot(exprs(gse21779_rma))
graphics::boxplot(exprs(gse21779_mas5))
```

4.10 Otros métodos posibles

Hemos visto dos métodos. Sin embargo, podemos considerar métodos distintos con el paquete [70, affy] ¿Qué metodos proporciona el paquete affy? Empezamos con los métodos de corrección de fondo. Las opciones son

```
bgcorrect.methods()
```

```
## [1] "bg.correct" "mas"        "none"
## [4] "rma"
```

Las opciones que nos muestra corresponden con:

none En la opción *none* no hacemos nada. Dejamos los valores originales.

mas Utiliza el procedimiento que hemos visto en MAS5.0.

RMA Utiliza el procedimiento visto en algoritmo RMA.

Nos ofrece los siguientes métodos de normalización.

```
normalize.methods(gse21779raw)
```

```
##  [1] "constant"           "contrasts"
##  [3] "invariantset"       "loess"
##  [5] "methods"            "qspline"
##  [7] "quantiles"          "quantiles.robust"
##  [9] "quantiles.probeset" "scaling"
```

En particular, para los chip de Affymetrix, tenemos distintas opciones para corregir los valores PM.

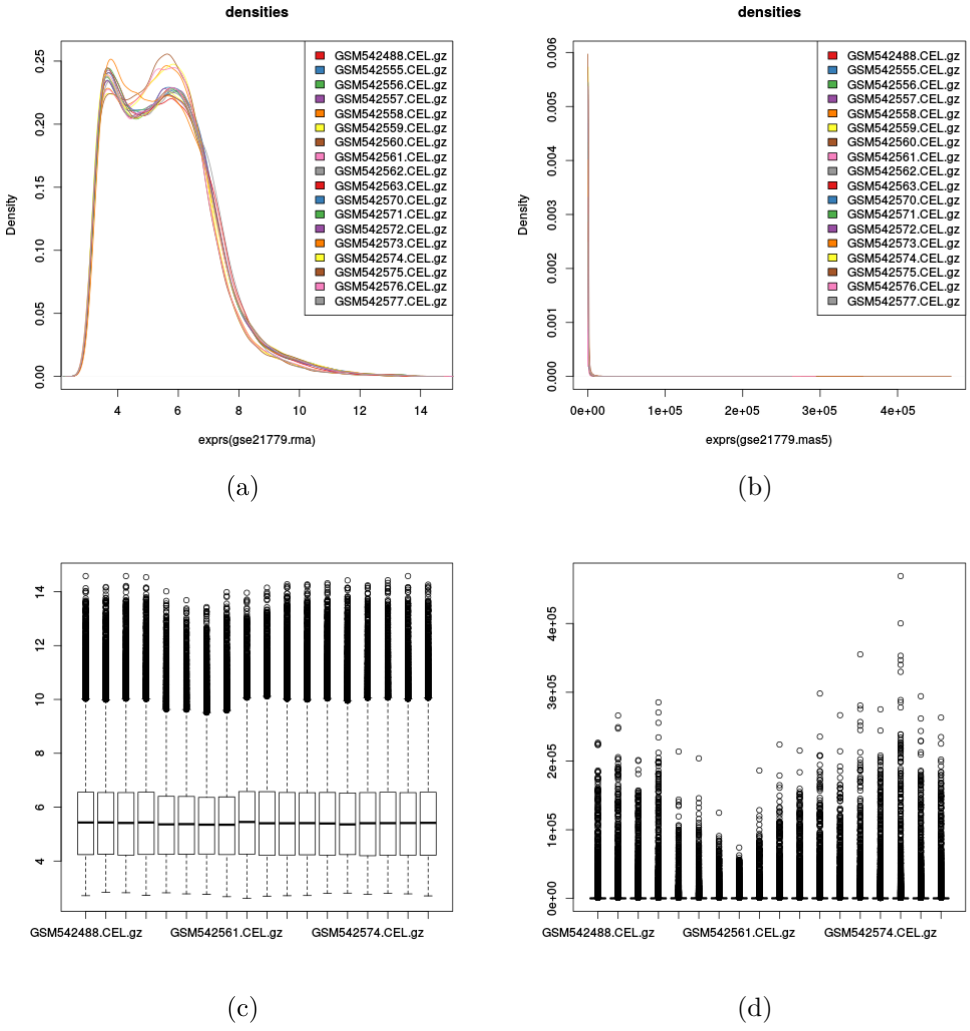


Figura 4.5: a) Estimadores kernel de densidad para los chips una vez hemos aplicado el procedimiento RMA a los datos gse21779. b) Lo mismo con el método MAS5. c) Boxplot para los niveles de expresión de los datos gse21779 con método RMA. d) Lo mismo que el punto (c) pero con el método MAS5.

```
pmcorrect.methods()

## [1] "mas"         "methods"     "pmonly"
## [4] "subtractmm"
```

Los métodos para resumir los valores de los conjuntos de sondas en un único valor de expresión del gen los podemos ver con el siguiente código.

```
express.summary.stat.methods()

## [1] "avgdiff"      "liwong"       "mas"
## [4] "medianpolish" "playerout"
```

La función *expresso* permite realizar los distintos pasos del preproceso. De hecho, la función `mas5` que hemos utilizado anteriormente no es más que una llamada a esta función.

```
mas5

## function (object, normalize = TRUE, sc = 500, analysis = "absolute",
##     ...)
## {
##     res <- expresso(object, bgcorrect.method = "mas", pmcorrect.method = "mas",
##         normalize = FALSE, summary.method = "mas", ...)
##     if (normalize)
##         res <- affy.scalevalue.exprSet(res, sc = sc, analysis = analysis)
##     return(res)
## }
## <bytecode: 0x55d7c7a0f278>
## <environment: namespace:affy>
```

Una explicación (bastante) detallada de cada una de las opciones la podemos encontrar en la viñeta *builtinMethods.pdf* del paquete [70, affy].

4.11 Cómo hacer un preprocesado sencillo a partir de datos de expresión

En la sección anterior hemos tratado cómo preprocesar datos partiendo de datos a nivel de sonda muy orientado a Affymetrix. Cuando conseguimos datos de expresión (por ejemplo de GEO) en ocasiones no tenemos los datos a nivel de sonda (los .CEL en el caso de Affymetrix GeneChip) sino que tenemos unos datos de expresión con un preproceso que no controlamos exactamente. El problema es que estos datos todavían necesitan un procesado para homogeneizarlos. ¿Cómo hacerlo de un modo sencillo? Una opción rápida la tenemos con la función limma::normalizeBetweenArrays.

Ejemplo 4.2 (GSE1397) *Veamos un ejemplo con los datos GSE1397. En § 5.10 hemos visto cómo bajarlos de GEO y cómo construir el ExpressionSet.*

```
load(paste(dirTamiData,"gse1397raw.rda",sep=""))
```

Son datos Affymetrix correspondientes al chip

```
annotation(gse1397raw)

## [1] "GPL96"
```

Supongamos que queremos aplicar una normalización de los cuantiles. [89] *Podemos hacer*

[89] Ver § 4.8.2 y [16].

```
library(limma)
exprs1 = normalizeBetweenArrays(exprs(gse1397raw),method = "quantile")
```

O bien podemos hacer que coincidan las medianas de los distintos arrays con

```
exprs1 = normalizeBetweenArrays(exprs(gse1397raw),method = "scale")
```

En la ayuda de limma::normalizeBetweenArrays podemos consultar otras opciones.

4.12 Otros métodos de normalización

El texto [128] es una excelente referencia sobre lo tratado en este tema y propone métodos adicionales. Mucho del material ha sido extraido de este libro.

En [92] se propone un método genérico de normalización partiendo de una matriz de expresión en donde cada columna corresponda a una muestra. En [92, página 2] se propone simplemente estimar las funciones de distribución de las distintas muestras y aplicando la transformada inversa a las funciones de distribución estimadas tendremos los nuevos datos. La normalización de percentiles utiliza esta idea y la distribución estimada es una uniforme sobre las medias de los percentiles observados. El método está implementado en https://github.com/mengqinxue/DBNorm.

Capítulo 5

Datos de microarrays

5.1 Introducción

Comentaremos los bancos de datos que utilizamos en el resto del curso. Son datos de expresión de genes obtenidos utilizando microarrays. Algunos van incorporados en los paquetes de R/Bioconductor como los datos golub (en [108]) o los datos ALL (en [80]). Se utilizan como ejemplos en muchos paquetes y en artículos. Es interesante analizarlos y conocerlos. También se muestra en este tema el uso de la clase Biobase::ExpressionSet. Esta clase permite el manejo de datos de expresión obtenidos con microarrays. Veremos también cómo bajar datos de repositorios públicos.

5.2 sample.ExpressionSet

Los datos Biobase::sample.ExpressionSet. El experimento tiene 26 muestras y 500 genes. Sobre las muestras conocemos tres variables (o covariables): sex, type (caso y control) y score. La última covariable no es categórica, es una covariable continua. Estos datos de expresión están almacenados en un objeto de clase ExpressionSet. Para poder utilizar esta estructura de datos hemos de cargar el paquete [52, Biobase].

```
library(Biobase)
```

Cargamos los datos.

```
data(sample.ExpressionSet)
```

Podemos ver ayuda sobre los mismos con

```
help(sample.ExpressionSet)
```

Las variables fenotípicas o metadatos que nos describen las muestras tienen las siguientes etiquetas.

```
varLabels(sample.ExpressionSet)

## [1] "sex"   "type"  "score"
```

Una variable dada, por ejemplo `type`, la tendremos con

```
sample.ExpressionSet$type
```

```
##  [1] Control Case    Control Case    Case
##  [6] Control Case    Case    Case    Control
## [11] Case    Control Case    Case    Case
## [16] Control Case    Control Case    Case
## [21] Control Control Control Control Case
## [26] Case
## Levels: Case Control
```

Y todos los datos fenotípicos se obtienen con

```
phenoData(sample.ExpressionSet)
```

```
## An object of class 'AnnotatedDataFrame'
##   sampleNames: A B ... Z (26 total)
##   varLabels: sex type score
##   varMetadata: labelDescription
```

La matriz de expresión se obtiene con (no la mostramos)

```
exprs(sample.ExpressionSet)
```

Para tener información sobre un objeto de clase ExpressionSet es recomendable consultar [65, capítulo 2]. Esto nos va a permitir trabajar con los datos de expresión y los metadatos que nos dan una descripción completa del experimento. En la clase ExpressionSet tendremos:

assayData Los datos de expresión de los distintos microarrays que componen toda la experimentación.

Metadatos En particular tendremos *phenoData* que nos dan información sobre las muestras utilizadas. Las características del chip vienen dadas en *featureData*. Las anotaciones de los genes las tendremos en *annotation*.

En la viñeta [52, Biobase] tenemos una buena descripción de la creación y manipulación de un ExpressionSet. Por ejemplo podemos saber la anotación de estos datos con

```
annotation(sample.ExpressionSet)
```

```
## [1] "hgu95av2"
```

Finalmente veamos una breve descriptiva de las covariables que nos proporcionan información sobre las muestras y que utilizaremos para ilustrar estas notas. Para las categóricas vemos una tabla de frecuencias absolutas.

```
table(sample.ExpressionSet$sex)
```

```
##
## Female   Male
##     11     15
```

```
table(sample.ExpressionSet$type)

##
##    Case Control
##      15      11
```

Y un resumen numérico de score.

```
summary(sample.ExpressionSet$score)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1000  0.3275  0.4150  0.5369  0.7650  0.9800
```

5.3 Datos golub

Una buena explicación de estos datos aparece en el manual de uso del paquete [108, multtest]. Estos datos fueron analizados por primera vez en [60]. En estas notas se utilizan en dos versiones. La primera es la incluida en el paquete [108]. La segunda corresponde a los datos incluidos en el paquete [61]. En el manual hablaremos de multtest::golub o bien golubEsets::golub_Train.[90]

Hay muestras correspondientes a dos grupos o condiciones: 27 muestras (las primeras 27 columnas) correspondientes a pacientes con leucemia linfoblástica aguda (ALL, la clase 0) y leucemia mieloide aguda (AML, clase 1). Para medir los niveles de expresión se utilizaron chips de alta densidad de oligonucleótidos Affymetrix para 6817 genes. Los datos los podemos obtener a partir del paquete [108] con

[90] Son unos datos antiguos pero que aparecen en numerosas publicaciones y viñetas de paquetes de R.

```
data(golub,package= "multtest")
```

Según se indica en el manual de [108, multtest] se realizó una selección previa de genes de carácter no específico (no se utiliza por tanto información de las dos condiciones que pretendemos diferenciar). Esta selección no específica consistió en los siguientes pasos. Se trabajó con los datos normalizados que se pueden encontrar en http://www.broadinstitute.org. Allí podemos encontrar una detallada explicación del protocolo experimental utilizado. Una vez obtenidos los datos normalizados se realizó la siguiente selección previa.

1. Una umbralización de los niveles de expresión. Valores por debajo de 100 se les asignó el valor 100. Los valores superiores a 16000 se les asignó el valor 16000.

2. Se calculó para cada gen el valor máximo y el valor mínimo de sus niveles de expresión para todas las muestras analizadas. Si denotamos por m_1 el mínimo y por m_2 el máximo entonces se eliminaron los genes que verifican

$$\frac{m_2}{m_1} \leq 5$$

   o bien que

$$m_2 - m_1 \leq 500$$

3. Se tomó el logaritmo en base 10 para los niveles de expresión de los genes que no fueron eliminados.

Finalmente se estandarizaron los niveles de expresión dentro de cada array. Es decir, se calculó para cada array la media y la desviación estándar de todos los niveles observados en este array. A los datos originales se les restó la media y se dividió por la desviación estándar. Estos son los datos finales con los que trabajamos.[1] Una vez hemos cargado los datos golub tenemos la siguiente información:

golub Es una matriz 3051×38 de niveles de expresión donde las 3051 filas corresponden a los genes y las 38 columnas corresponden a las muestras. Podemos ver su primera fila:

```
golub[1,]

##  [1] -1.45769 -1.39420 -1.42779 -1.40715 -1.42668
##  [6] -1.21719 -1.37386 -1.36832 -1.47649 -1.21583
## [11] -1.28137 -1.03209 -1.36149 -1.39979  0.17628
## [16] -1.40095 -1.56783 -1.20466 -1.24482 -1.60767
## [21] -1.06221 -1.12665 -1.20963 -1.48332 -1.25268
## [26] -1.27619 -1.23051 -1.43337 -1.08902 -1.29865
## [31] -1.26183 -1.44434  1.10147 -1.34158 -1.22961
## [36] -0.75919  0.84905 -0.66465
```

golub.gnames Es una matriz 3051×3 con identificadores de los genes. Veamos la información de la primera fila correspondiente al primer gen.

```
golub.gnames[1,]

## [1] "36"
## [2] "AFFX-HUMISGF3A/M97935_MA_at (endogenous control)"
## [3] "AFFX-HUMISGF3A/M97935_MA_at"
```

golub.cl Es un vector de longitud 38 con la clasificación del tipo de leucemia.

```
golub.cl

##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [24] 0 0 0 0 1 1 1 1 1 1 1 1 1 1
```

Con frecuencia, y con objeto de mejorar la presentación de los resultados ejecutaremos la siguiente línea de código.

```
golub.cl = factor(golub.cl,levels= 0:1,labels=c("ALL","AML"))
```

Si volvemos a ver el contenido de golub.cl veremos el cambio producido.

[1] Es importante darse cuenta de la enorme cantidad de transformaciones de la información original que estamos haciendo y que, por lo tanto, afecta de un modo fundamental las conclusiones que obtengamos en cualquier tratamiento estadístico posterior. Y lo peor: no sabemos de un modo claro cómo afectan estos resultados.

```
golub.cl

##  [1] ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL
## [12] ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL
## [23] ALL ALL ALL ALL ALL AML AML AML AML AML AML
## [34] AML AML AML AML AML
## Levels: ALL AML
```

[91]

En el paquete [61] tenemos los datos originales. Tanto las muestras que se utilizaron para entrenar el procedimiento como las que se utilizaron para evaluarlo.

[91] El script de R con todo el preprocesado descrito podemos encontrarlo en http://svitsrv25.epfl.ch/R-doc/library/multtest/doc/golub.R y en la documentación del paquete [108].

5.4 Datos ALL

Para una versión ampliada de esta sección podemos consultar [65, capítulo 1]. Los datos ALL son microarrays de 128 individuos distintos con leucemia linfoblástica aguda (ALL). De estos individuos 95 corresponden a leucemia linfoblástica precursora aguda de células B y y 33 son leucemia linfoblástica precursora aguda de células T. Son enfermedades bastante distintas y por ello se consideran por separado. Habitualmente trabajaremos con las muestras de leucemia linfoblástica precursora aguda de células B. Los datos han sido preprocesados utilizando el método RMA (robust multichip average) implementado en el paquete [70, affy] con la función affy::rma y están almacenados en forma de un Biobase::ExpressionSet. Empezamos cargando los datos.

```
library(Biobase)
library(ALL)
data(ALL)
```

En lo que sigue no vamos a utilizar todas las muestras.

Un subconjunto de muestras de interés con dos grupos son los grupos de tumores de células B que tienen la mutación BCR/ABL y los tumores de las células B sin ninguna anormalidad citogenética. Veamos cómo seleccionar estas muestras. En primer lugar seleccionamos las muestras de células B.

```
bcell = grep("^B",as.character(ALL$BT))
```

Y ahora seleccionamos las muestras correspondientes a los tipos moleculares BCR/ABL o NEG.

```
types = c("NEG","BCR/ABL")
moltyp = which(as.character(ALL$mol.biol) %in% types)
```

Ahora combinamos ambas selecciones para quedarnos con los tumores de las células B y que tienen o bien la translocación BCR/ABL o bien no tienen ninguna de las anormalidades moleculares evaluadas.

```
bcrneg = ALL[,intersect(bcell,moltyp)]
```

Habitualmente haremos uso de los datos ALL realizando previamente esta selección. Remitiremos a esta sección para consultar el código anterior.

5.5 Construyendo un ExpressionSet

Supongamos (y no es mucho suponer) que tenemos los datos de expresión en un fichero, en otro fichero tenemos la descripción de las distintas muestras que hemos analizado, en otro fichero (o en la red) tenemos información sobre los genes (o sondas) que tenemos en las filas de la matriz de expresión. Tenemos información sobre las distintas muestras que componen toda la experimentación, bajo qué condiciones se consiguieron. Lo que podemos llamar la descripción fenotípica de las muestras. Y finalmente sabemos cosas sobre toda la experimentación realizada: autores, publicación principal que se derivó, objetivo de la experimentación. Y, como es habitual, lo tenemos todo por separado, en distintos ficheros. La clase Biobase::ExpressionSet tiene como objetivo tenerlo todo contenido en un mismo objeto. En esta sección vemos cómo construir este ExpressionSet a partir de la información dispersa. En ocasiones esto nos puede pasar si bajamos la información de alguna publicación en la que nos aparece como información suplementaria.

Es probable que en nuestro trabajo no tengamos que realizar todo el proceso que indicamos a continuación pero con mucha probabilidad al menos una parte sí.

ExpressionSet

Es una estructura de datos que nos permite mantener en un solo objeto toda la información que relativa a un experimento con microarrays.

Para poder utilizar esta clase necesitamos el paquete [52, Biobase]. Lo cargamos (la clase y los métodos que luego vamos a utilizar).

```
pacman::p_load(Biobase)
```

¿Qué necesitamos?

Supongamos que tenemos la información por separado y queremos construir un ExpressionSet. ¿Qué datos nos describen un experimento genómico de alto rendimiento? Podemos distinguir la siguiente información:

1. Los datos del ensayo (los niveles de expresión).
2. Datos sobre las muestras que podemos llamar de un modo genérico *metadatos fenotípicos*.
3. Anotaciones sobre las sondas u otros metadatos.
4. Una descripción del experimento.

Vamos a construir un ExpressionSet construyendo cada una de las componentes a partir de unos datos.[2]

Datos del ensayo

Son los valores de expresión posiblemente preprocesados. Es una matriz con N filas y con n columnas. El número N de filas corresponde con el número de características (features) del chip y n con el número de muestras. Leemos los datos (posiblemente con `read.table` o `read.csv`. Nos han pasado los datos.[92] Normalmente las funciones `utils::read.table()` o `utils::read.csv()` nos suelen resolver el problema.

[92] Como Dios quiere y nosotros hemos de averiguar.

```
pacman::p_load("tamidata")
finput = system.file("extdata","n06_expr.txt",package="tamidata")
exprs0 = read.table(file = finput,header = TRUE,sep="\t")
```

Cuando usamos estas funciones nos devuelve un `data.frame` como podemos comprobar.

```
class(exprs0)

## [1] "data.frame"
```

En lo que sigue necesitamos que sea una matriz. Lo hacemos (por cierto el -1 nos sirve para quitar la primera columna de la matriz donde teníamos los nombres de las sondas).

```
exprs0 = as.matrix(exprs0[,-1])
```

Y podemos comprobar que ya lo tenemos.

```
class(exprs0)

## [1] "matrix" "array"
```

¿Cuántas filas (features) y columnas (samples) tenemos?

```
dim(exprs0)

## [1] 22215    54
```

Los nombres de las columnas son

```
colnames(exprs0)

##  [1] "X600MPE"    "AU565"      "BT474"
##  [4] "BT483"      "CAMA1"      "DU4475"
##  [7] "HCC202"     "HCC1428"    "HCC1007"
## [10] "HCC2185"    "LY2"        "MCF7"
## [13] "MDAMB415"   "MDAMB175"   "MDAMB361"
## [16] "MDAMB435"   "MDAMB134"   "SUM185PE"
```

[2] Los datos que estamos usando han sido amablemente proporcionados por Joan Climent y están ligeramente modificados respecto de los publicados originalmente en [100].

```
## [19] "SUM225CWN"  "SUM44PE"    "SKBR3"
## [22] "T47D"       "UACC812"    "ZR75B"
## [25] "ZR751"      "ZR7530"     "SUM52PE"
## [28] "BT20"       "HCC1937"    "HCC70"
## [31] "HCC1187"    "HCC1008"    "HCC1599"
## [34] "HCC3153"    "HCC1569"    "HCC2157"
## [37] "HCC1954"    "HCC38"      "HCC1500"
## [40] "HCC1143"    "MCF12A"     "MCF10A"
## [43] "MDAMB468"   "SUM149PT"   "SUM190PT"
## [46] "BT549"      "HS578T"     "HBL100"
## [49] "MDAMB231"   "MDAMB435_2" "MDAMB157"
## [52] "MDAMB436"   "SUM159PT"   "SUM1315"
```

Y podemos ver

```
head(exprs0,n=1)

##       X600MPE    AU565    BT474    BT483   CAMA1
## [1,] 10.11474 10.42933 9.445266 10.16027 10.0863
##       DU4475   HCC202  HCC1428  HCC1007  HCC2185
## [1,] 8.79988 10.32634 9.279211 9.620473 9.620686
##           LY2    MCF7 MDAMB415 MDAMB175 MDAMB361
## [1,] 9.471435 9.87178 10.07334 9.736757 10.20135
##      MDAMB435 MDAMB134 SUM185PE SUM225CWN
## [1,] 9.438713 9.355205 10.18208  10.35122
##       SUM44PE   SKBR3     T47D  UACC812    ZR75B
## [1,] 10.28434 9.58074 9.968553 10.68766 9.624339
##         ZR751   ZR7530  SUM52PE    BT20  HCC1937
## [1,] 9.872964 9.843985 9.584688 9.52413 9.881727
##         HCC70  HCC1187  HCC1008  HCC1599  HCC3153
## [1,] 10.53008 10.18941 10.14184 9.933858 9.920917
##       HCC1569 HCC2157  HCC1954    HCC38  HCC1500
## [1,] 9.009182 9.30533 8.945598 9.207611 9.857016
##       HCC1143   MCF12A   MCF10A MDAMB468 SUM149PT
## [1,] 10.35362 9.458165 8.582511 10.12283 9.849195
##      SUM190PT    BT549   HS578T   HBL100 MDAMB231
## [1,] 9.497212 8.225951 7.960181 8.137387 7.625879
##      MDAMB435_2 MDAMB157 MDAMB436 SUM159PT
## [1,]   8.038661 8.320586 7.414636 7.962508
##       SUM1315
## [1,] 7.512639
```

Vamos a etiquetar las filas con los identificadores de las sondas.

```
finput = system.file("extdata","n06_gene_attributes.txt",package="tamidata")
rn = read.table(file = finput,header = TRUE,sep="\t")
rownames(exprs0) = rn[,1]
```

Datos fenotípicos

Es la información que tenemos de las distintas muestras. Será una matriz de datos (un data.frame) con tantas filas como muestras, n, y con tantas columnas como variables tenemos. Estas variables que nos

describen las muestras podemos llamarlas covariables (una denominación muy estadística).

En nuestro caso los datos fenotípicos los tenemos en el fichero `sample_attributes.txt`. Los leemos.

```
finput = system.file("extdata","n06_sample_attributes.txt",
    package="tamidata")
pd0 = read.table(finput,sep = "\t",header=TRUE)
```

Podemos ver el número de muestras (que ha de coincidir con el número de columnas de la matriz de expresión) y de covariables.

```
dim(pd0)

## [1] 54 14
```

Es **necesario** que los nombres de las filas de `pd0` coincidan con los nombres de las columnas de `exprs0` ya que nos están dando covariables relativas a estas muestras.

```
rownames(pd0) = colnames(exprs0)
```

Esto es fundamental. Si no coinciden los nombres de las columnas de la matriz de expresión con los nombres de las filas de la matriz de datos de las covariables no podremos construir el `ExpressionSet`. Y ahora es conveniente un pequeño resumen descriptivo de las covariables.

```
summary(pd0)

##   IDENTIFIER          sample_no
##  Length:54          Min.   : 1.00
##  Class :character   1st Qu.:14.25
##  Mode  :character   Median :27.50
##                     Mean   :27.50
##                     3rd Qu.:40.75
##                     Max.   :54.00
##
##      Type               in_CGH           ALL
##  Length:54          Min.   :0.000   Min.   :1
##  Class :character   1st Qu.:1.000   1st Qu.:1
##  Mode  :character   Median :1.000   Median :1
##                     Mean   :0.963   Mean   :1
##                     3rd Qu.:1.000   3rd Qu.:1
##                     Max.   :1.000   Max.   :1
##
##   CDH1_meth             SFRP1
##  Length:54          Min.   :2.975
##  Class :character   1st Qu.:3.160
##  Mode  :character   Median :3.293
##                     Mean   :3.701
##                     3rd Qu.:4.004
##                     Max.   :7.780
##                     NA's   :1
##      HER2               KIT
```

```
##  Length:54          Length:54
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
##
##   known.p53             BRCA
##  Length:54          Length:54
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
##
##      order        adherence
##  Min.   : 1.00   Length:54
##  1st Qu.:14.00   Class :character
##  Median :27.00   Mode  :character
##  Mean   :27.23
##  3rd Qu.:40.00
##  Max.   :54.00
##  NA's   :1
##       per3
##  Min.   :4.000
##  1st Qu.:4.000
##  Median :5.000
##  Mean   :5.452
##  3rd Qu.:5.000
##  Max.   :9.000
##  NA's   :23
```

Tenemos covariables categóricas (en R se les llama de clase `factor`) y covariables numéricas. Con las covariables categóricas tenemos las frecuencias mientras que con las numéricas nos da mínimo, primer cuartil (o percentil de orden 25 %), mediana y media, tercer cuartil (o percentil de orden 75 %) y máximo. Podemos ver los tipos (o clases) de cada columna con `sapply()`.

```
sapply(pd0,class)
```

```
##  IDENTIFIER   sample_no        Type      in_CGH
## "character"   "integer" "character"   "integer"
##         ALL   CDH1_meth       SFRP1        HER2
##   "integer" "character"   "numeric" "character"
##         KIT   known.p53        BRCA       order
## "character" "character" "character"   "integer"
##   adherence        per3
## "character"   "integer"
```

Podemos tener información adicional sobre las covariables que nos describen las muestras. Por ejemplo, un nombre de variable como `known.p53` puede no significar demasiado. Pero si una etiqueta como "Presencia de p53"[3] ¿Cómo podemos añadir esta información?

[3] No es un ejemplo muy lucidor.

```
metadatos = data.frame(labelDescription = c("Identificador",
"Número de muestra", "Tipo","En CGH","ALL","Método CDH1","SFRP1",
"HER2","KIT","Presencia de p53","BRCA","Orden","Adherencia",
"per3"),row.names=colnames(pd0))
```

Notemos que la columna labelDescription debe de estar presente. Utilizamos la clase Biobase::AnnotatedDataFrame que nos permite tener los valores de las covariables y los metadatos que acabamos de introducir. Lo podemos hacer del siguiente modo.

```
datosfenotipo = new("AnnotatedDataFrame", data = pd0,
                    varMetadata = metadatos)
```

O bien con

```
datosfenotipo = AnnotatedDataFrame(data = pd0,varMetadata = metadatos)
```

Una vez tenemos este `AnnotatedDataFrame` podemos ver los nombres de las muestras

```
sampleNames(datosfenotipo)
```

```
##  [1] "X600MPE"    "AU565"      "BT474"
##  [4] "BT483"      "CAMA1"      "DU4475"
##  [7] "HCC202"     "HCC1428"    "HCC1007"
## [10] "HCC2185"    "LY2"        "MCF7"
## [13] "MDAMB415"   "MDAMB175"   "MDAMB361"
## [16] "MDAMB435"   "MDAMB134"   "SUM185PE"
## [19] "SUM225CWN"  "SUM44PE"    "SKBR3"
## [22] "T47D"       "UACC812"    "ZR75B"
## [25] "ZR751"      "ZR7530"     "SUM52PE"
## [28] "BT20"       "HCC1937"    "HCC70"
## [31] "HCC1187"    "HCC1008"    "HCC1599"
## [34] "HCC3153"    "HCC1569"    "HCC2157"
## [37] "HCC1954"    "HCC38"      "HCC1500"
## [40] "HCC1143"    "MCF12A"     "MCF10A"
## [43] "MDAMB468"   "SUM149PT"   "SUM190PT"
## [46] "BT549"      "HS578T"     "HBL100"
## [49] "MDAMB231"   "MDAMB435_2" "MDAMB157"
## [52] "MDAMB436"   "SUM159PT"   "SUM1315"
```

O también ver los valores de las covariables (mostramos los primeros solamente)

```
head(pData(datosfenotipo))
```

```
##         IDENTIFIER sample_no Type in_CGH ALL
## X600MPE     600MPE         1    L      1   1
## AU565        AU565         2    L      1   1
## BT474        BT474         3    L      1   1
## BT483        BT483         4    L      1   1
## CAMA1        CAMA1         5    L      1   1
## DU4475      DU4475         6    L      1   1
##         CDH1_meth   SFRP1 HER2 KIT known.p53 BRCA
```

```
## X600MPE        no 3.40987 <NA>  NO        NO <NA>
## AU565        <NA> 3.16902 <NA>  NO       YES <NA>
## BT474          no 3.16737  YES  NO        NO loss
## BT483          no 2.97456   NO  NO        NO   WT
## CAMA1          no 3.16856 <NA>  NO        NO   WT
## DU4475         no 3.06074 <NA>  NO        NO   WT
##         order adherence per3
## X600MPE     1      <NA>    9
## AU565       2      <NA>    4
## BT474       3      <NA>   NA
## BT483       4      <NA>   NA
## CAMA1       5      <NA>   NA
## DU4475      6      <NA>   NA
```

Anotaciones y datos de las características

Sin duda este es el punto fundamental, la descripción de información sobre las filas de la matriz de expresión. Estas filas (features) corresponden con sondas. De hecho, distintas sondas corresponden con un mismo gen. Esta información depende del chip que utilizamos. Por ello distintos experimentos pueden corresponder a un mismo chip o instrumento. Es conveniente consultar http://www.bioconductor.org/packages/2.11/bioc/html/annotate.html. La información de cada chip aparece en un paquete de metadatos del mismo. En particular, nos proporcionan el nombre del gen, el símbolo y la localización en el cromosoma. Otros paquetes proporcionan información de GO o KEGG.

Descripción del experimento

Los experimentos los hace la gente y hay que incluir esta información en los datos.

```
datosexperimento = new('MIAME',name='Neve et alt.',lab='Varios',
    contact ='rmneve@lbl.gov',
    title = 'A collection of breast cancer cell lines for the study of
functionally distinct cancer subtypes',abstract = 'An example ExpressionSet',
    url = '10.1016/j.ccr.2006.10.008',
    other = list(notes = 'Creado a partir  de ficheros de texto'))
```

El formato MIAME (Minimum information about a microarray experiment) puede consultarse en http://fged.org/projects/miame/.

Y montamos las piezas

Ejemplo de ExpressionSet a partir de los distintos elementos que hemos construido.[93]

[93] Esto es un poco como el Mecano.

```
neve06  = new("ExpressionSet",exprs=exprs0,phenoData = datosfenotipo,
              experimentData = datosexperimento,
              annotation = "hgu95av2")
```

Equivalentemente podemos usar

```
neve06  = ExpressionSet(assayData=exprs0,phenoData = datosfenotipo,
                        experimentData = datosexperimento,
                        annotation = "hgu95av2")
```

Guardamos este ExpressionSet para uso posterior.

```
save(neve06,file="neve06.rda")
```

Todo lo hecho en esta sección se puede hacer de un modo más simple con EnrichmentBrowser::read.eset.

5.6 Un experimento con levadura

Los datos que comentamos en esta sección son un experimento en donde vamos a considerar dos tipos de células en levadura: salvaje y mutada.

Los datos los tenemos en un `ExpressionSet` que leemos.

```
data(gse6647,package="tamidata")
```

El número de genes y muestras es

```
dim(gse6647)

## Features  Samples
##     6103        8
```

Podemos ver los datos fenotípicos.

```
head(pData(gse6647))

##                    type
## GSM153907.CEL.gz     wt
## GSM153908.CEL.gz edc3D
## GSM153909.CEL.gz     wt
## GSM153910.CEL.gz edc3D
## GSM153911.CEL.gz     wt
## GSM153912.CEL.gz edc3D
```

Los datos han sido normalizados utilizando el método RMA. En la figura 5.2(a) mostramos las densidades estimadas.

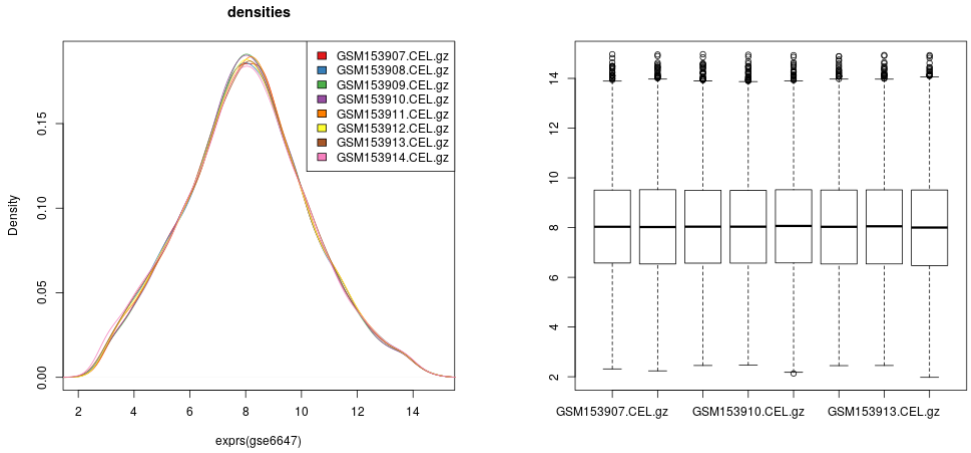
```
geneplotter::multidensity(exprs(gse6647))
```

En la figura 5.2(b) tenemos los diagramas de cajas.

```
boxplot.matrix(exprs(gse6647))
```

Vamos a reproducir los dibujos de la figura 5.2 utilizando el paquete [145, ggplot2].[94] El código es el que sigue y los tenemos en la figura 5.2.

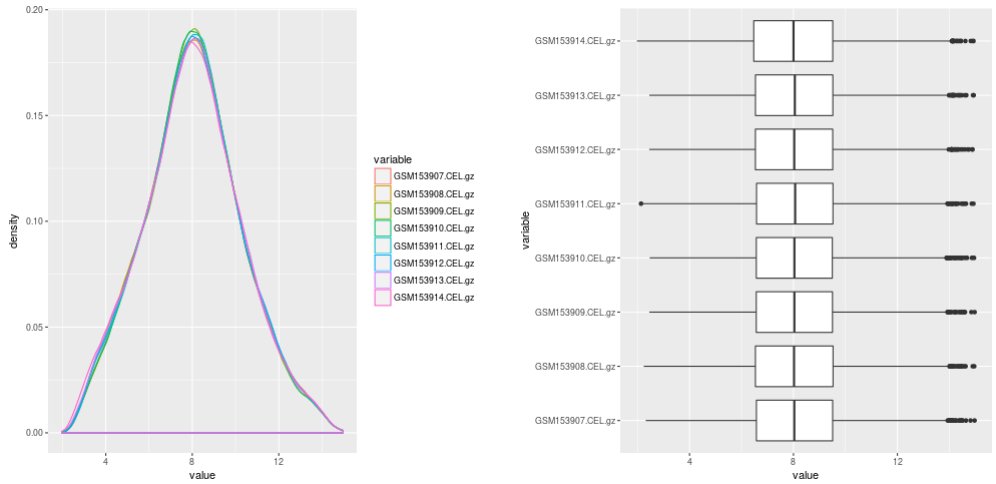
[94] Sobre el paquete [141, reshape] es interesante consultar [142].



(a) (b)

Figura 5.1: Datos gse6647: estimadores de densidad y diagramas de cajas.

```
library(reshape);library(ggplot2)
df = data.frame(gene = featureNames(gse6647),exprs(gse6647))
df1 = melt(df,id=c("gene"))
ggplot(df1,aes(x=value,colour=variable,group=variable)) +
    geom_density(kernel = "epanechnikov",fill=NA) ## Estimadores densidad
ggplot(df1,aes(x=variable,y = value)) + geom_boxplot() +
    coord_flip()
```



(a) (b)

Figura 5.2: Datos gse6647: estimadores de densidad y diagramas de cajas utilizando [145, ggplot2].

5.7 De cómo utilizar un ExpressionSet

En esta sección pretendemos mostrar cómo utilizar funciones que manejan ExpressionSet. Vamos a utilizar los datos ALL que aparecen en el paquete [80, ALL]. Cargamos el paquete.

```
library(ALL)
```

Podemos ver que es un resumen de la información que tenemos en ALL.

```
ALL
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 12625 features, 128 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 01005 01010 ... LAL4 (128
##     total)
##   varLabels: cod diagnosis ... date last
##     seen (21 total)
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
##   pubMedIds: 14684422 16243790
## Annotation: hgu95av2
```

Podemos ver los niveles de expresión (en filas tenemos sondas y en columnas muestras). Los niveles de expresión han sido preprocesados y aparecen el logaritmo en base 2 de este valor ya preprocesado. Se obtienen (y no lo mostramos) con

```
exprs(ALL)
```

Los datos fenotípicos los tendremos con (tampoco los mostramos)

```
pData(ALL)
```

¿Cuál fue chip que se utilizó para obtener estos datos?

```
annotation(ALL)
```

```
## [1] "hgu95av2"
```

Con `pData(ALL)` hemos visto los nombres (y los valores) de las covariables que nos describen las distintas muestras. Solamente los nombres los podemos ver con

```
names(pData(ALL))
```

```
##  [1] "cod"            "diagnosis"
##  [3] "sex"            "age"
##  [5] "BT"             "remission"
```

```
##  [7] "CR"             "date.cr"
##  [9] "t(4;11)"        "t(9;22)"
## [11] "cyto.normal"    "citog"
## [13] "mol.biol"       "fusion protein"
## [15] "mdr"            "kinet"
## [17] "ccr"            "relapse"
## [19] "transplant"     "f.u"
## [21] "date last seen"
```

Y si simplemente queremos ver los valores de la variable BT podemos hacerlo con

```
ALL$BT
```

```
##   [1] B2 B2 B4 B1 B2 B1 B1 B1 B2 B2 B3 B3 B3 B2 B3
##  [16] B  B2 B3 B2 B3 B2 B2 B2 B1 B1 B2 B1 B2 B1 B2
##  [31] B  B  B2 B2 B2 B1 B2 B2 B2 B2 B2 B4 B4 B2 B2
##  [46] B2 B4 B2 B1 B2 B2 B3 B4 B3 B3 B3 B4 B3 B3 B1
##  [61] B1 B1 B1 B3 B3 B3 B3 B3 B3 B3 B3 B1 B3 B1 B4
##  [76] B2 B2 B1 B3 B4 B4 B2 B2 B3 B4 B4 B4 B1 B2 B2
##  [91] B2 B1 B2 B  B  T  T3 T2 T2 T3 T2 T  T4 T2 T3
## [106] T3 T  T2 T3 T2 T2 T2 T1 T4 T  T2 T3 T2 T2 T2
## [121] T2 T3 T3 T3 T2 T3 T2 T
## Levels: B B1 B2 B3 B4 T T1 T2 T3 T4
```

Los datos de expressión aparecen en una matriz con filas (features que suelen corresponder a sondas) y con columnas (que corresponden con muestras). Vamos a seleccionar una parte de las muestras. En concreto, vamos a considerar las muestras tales que la variable fenotípica `mol.bio` toma el valor `NEG`. Determinamos las columnas.

```
selcol = ALL$mol.biol == "NEG"
```

Y nos quedamos con esa muestras.

```
ALL[,selcol]
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 12625 features, 74 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 01010 04007 ... LAL4 (74
##     total)
##   varLabels: cod diagnosis ... date last
##     seen (21 total)
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
##   pubMedIds: 14684422 16243790
## Annotation: hgu95av2
```

Lo importante es ver que todo el ExpressionSet tiene menos muestras y también se han modificado (eliminando) los datos de expresión y los datos fenotípicos.

5.8 NCBI GEO

GEO es el NCBI Gene Expression Omnibus. Su dirección es http://www.ncbi.nlm.nih.gov/geo/. Es un repositorio público con datos experimentales de alto rendimiento. Tenemos experimentos basados en microarrays de uno o dos canales que miden mRNA, DNA genómico, presencia de proteinas. También hay otras técnicas no basadas de arrays como análisis serial de expresión de genes (SAGE), datos proteómicos obtenidos son espectrometría de masas o datos de secuenciación de alto rendimiento. Hay cuatro tipos de entidades básicas en GEO: Sample, Platform, Series, DataSet. En esta sección mostramos cómo obtener datos de esta gran base de datos pública. Los datos que nos bajamos los utilizaremos en los siguientes temas. Cuando accedemos a una entrada de GEO podemos ver el enlace **Analyze with GEO2R**. Si accedemos a este enlace podemos ver que nos ofrece algunos análisis de los datos utilizando R/Bioconductor. Estas notas son una versión muy extendida de estos análisis.

5.9 GSE21779

Hemos elegido los datos que en dicha base de datos tienen el identificador GSE21779. Podemos buscarlos utilizando la página web http://www.ncbi.nlm.nih.gov/geo/ o bien podemos acceder directamente a esta dirección http://www.ncbi.nlm.nih.gov/sites/GDSbrowser?acc=GDS4128. Bajamos los ficheros CEL (están en formato comprimido gzip).

Sabiendo el identificador de los datos y que tienen los datos de expresión originales los podemos bajar del siguiente modo. Primero cargamos el paquete [37, GEOquery].

```
library(GEOquery)
```

Y sabiendo el identificador podemos bajar los datos.

```
gcel = getGEOSuppFiles("GSE21779")
```

Podemos ver que los coloca en un subdirectorio al que denomina `GSE21779`.[95] Cambiamos al directorio donde tenemos los datos.

[95] A gusto del consumidor. Posiblemente para bajar un simple banco de datos lo mejor es utilizar el navegador.

```
setwd("GSE21779/")
```

Descomprimimos.

```
system("tar xvf GSE21779_RAW.tar")
```

Cargamos el paquete [70, affy].

```
library(affy)
```

Y leemos todos los datos.

```
gse21779 = ReadAffy()
```

Guardamos estos datos de expresión en un fichero externo.

```
save(gse21779,file = "gse21779.rda")
```

```
save(gse21779,file=paste(dirTamiData,"gse21779.rda",sep=""))
```

Otra opción es bajar los mismos datos de ArrayExpress. Su código en esta base de datos es `E-GEOD-21779`. Lo hacemos con el siguiente código.

```
library(ArrayExpress)
geod21779 = ArrayExpress("E-GEOD-21779")
```

La ventaja es que directamente nos devuelve un AffyBatch. Normalizamos los datos mediante el método RMA.[96]

[96] § 4.8.

```
geod21779.rma = rma(geod21779)
```

```
save(geod21779.rma,file=paste(dirTamiData,"geod21779.rma.rda",sep=""))
```

5.10 GSE1397

[97] El uso de estos datos y parte del análisis aparecen en un documento que se puede encontrar en esta dirección. También he utilizado parte del análisis que propone Rodrigo Santamaría.

[97] El síndrome de Down es una enfermedad causada por la aparición de un copia extra total o parcial del cromosoma 21. Analizaremos si los genes de este cromosoma muestra una sobre expresión. Además se verá que solamente es específica de estos genes y no de otros que aparecen en otros cromosomas. Los datos se pueden obtener de GEO y el identificador del experimento es GSE1397. Los resultados que se obtienen son realmente bonitos e interesantes. Los datos a nivel de sonda (raw data) no están disponibles. Nos hemos de traer unos datos preprocesados. En el experimento se utilizaron arrays Affymetrix GeneChip U133A. Los obtenemos con

```
pacman::p_load("Biobase","GEOquery")
gse1397raw = getGEO("GSE1397")[[1]]
```

La anotación de estos datos es `GPL96` que (podemos comprobarlo entrando en GEO) corresponde con `Affymetrix Human Genome U133` chip hgu133a. El paquete que contiene sus datos de anotación es [26, hgu133a.db]. Cambiamos la anotación.

```
annotation(gse1397raw) = "hgu133a"
```

Los datos fenotípicos de las muestras los podemos ver con (no mostramos)

```
pData(gse1397raw)
```

Podemos ver las etiquetas de las covariables asociadas a las muestras con

```
varLabels(gse1397raw)
```

Fijémonos en la variable `type` que nos indica la alteración. En concreto puede ser trisomía 21 (síndrome de Down), trisomía 13 o euploide (ploidez normal) y el tejido puede ser cerebro, cerebelo, astrocito y corazón. Los valores de esta covariable la podemos ver con

```
head(pData(gse1397raw)[,"title"])
```

La matriz de expresión la tenemos (y no la mostramos) con

```
exprs(gse1397raw)
```

Vamos a seleccionar aquellas muestras que corresponden a cerebro o cerebelo y que son o bien TS21 o euploides.

```
(ts21=c(grep("T.*21.*cerebrum", as.character(pData(gse1397raw)[,"title"])),
  grep("TS21.*cerebellum", as.character(pData(gse1397raw)[,"title"]))))
(eu=c(grep("Euploid.*cerebrum)", as.character(pData(gse1397raw)[,"title"])),
  grep("Euploid.*[Cc]erebellum", as.character(pData(gse1397raw)[,"title"]))))
```

Efectivamente tenemos las columnas con las muestras que buscábamos

```
pData(gse1397raw)[eu,"title"]
```

```
pData(gse1397raw)[ts21,"title"]
```

Y ahora seleccionamos las muestras.

```
gse1397raw = gse1397raw[,c(eu,ts21)]
```

Modificamos los datos fenotípicos.

```
tissue = factor(rep(c(1,2,1,2),c(4,3,4,3)),levels=1:2,
  labels=c("Cerebrum","Cerebellum"))
type = factor(c(rep(1,7),rep(2,7)),levels=1:2,labels=c("Euploid","TS21"))
GROUP = as.numeric(type) - 1
pData(gse1397raw)  = data.frame(tissue,type)
```

Guardamos los datos.

```
save(gse1397raw,file="gse1397raw.rda")
```

[98] Vamos a mostrar los estimadores kernel de la densidad en su escala original. Primero cargamos el paquete [17, affyPLM] que, entre otras cosas, nos permite obtener estos estimadores con facilidad.

```
library(affyPLM)
```

No tenemos los ficheros .CEL que necesitamos para preprocesar los datos. Por ello recurrimos a la función `normalizeBetweenArrays()` del paquete [126, limma].

[98] El análisis que acabamos de ver desde bajar los datos hasta obtener el ExpressionSet lo podemos encontrar en http://www.uv.es/ayala/docencia/tami/Rmd/gse1397.html.

```
library(limma)
gse1397 = gse1397raw
exprs(gse1397) = normalizeBetweenArrays(exprs(gse1397raw))
```

```
df = data.frame(gene = featureNames(gse1397),exprs(gse1397))
df1 = melt(df,id=c("gene"))
png(paste(dirTamiFigures,"gse1397normden.png",sep=""))
ggplot(df1,aes(x=value,colour=variable,group=variable)) +
    geom_density(kernel = "epanechnikov",fill=NA) + xlim(0,1000)
dev.off()
```

[99] gse1397.

[99]

Los autores introdujeron información adicional sobre las filas (genes) que podemos ver (no lo mostramos) con

```
fData(gse1397raw)
```

Es un `data.frame` que contiene las siguientes variables sobre los genes.

```
names(fData(gse1397raw))
```

```
##  [1] "ID"
##  [2] "GB_ACC"
##  [3] "SPOT_ID"
##  [4] "Species Scientific Name"
##  [5] "Annotation Date"
##  [6] "Sequence Type"
##  [7] "Sequence Source"
##  [8] "Target Description"
##  [9] "Representative Public ID"
## [10] "Gene Title"
## [11] "Gene Symbol"
## [12] "ENTREZ_GENE_ID"
## [13] "RefSeq Transcript ID"
## [14] "Gene Ontology Biological Process"
## [15] "Gene Ontology Cellular Component"
## [16] "Gene Ontology Molecular Function"
```

5.11 Datos GSE20986

[4]

Empezamos obteniendo los datos. Tenemos los datos a nivel de sonda de modo que podemos realizar nuestro propio preprocesado. Una explicación detallada del protocolo experimental se puede ver en GEO consultando la información de una cualquiera de las muestras, por ejemplo, en http://www.ncbi.nlm.nih.gov/geo/query/

[4] El análisis de estos datos lo sugirió un estudio que se puede encontrar en http://bioinformatics.knowledgeblog.org/2011/06/20/analysing-microarray-data-in-bioconductor/. En lo que sigue utilizaremos parte del análisis que se propone allí. La referencia original del estudio es [22].

acc.cgi?acc=GSM524662. Podemos ver en esta página que el preprocesado de los datos para pasar de los datos a nivel de sonda a datos de expresión se realizó utilizando el método GC-RMA (en concreto utilizaron la función gcrma::gcrma del paquete [149, gcrma].

La plataforma GEO[100] es GPL570 que es su código para Affymetrix Human Genome U133 Plus 2.0 Array. Empezamos cargando paquetes necesarios.

[100] El código que le dan en GEO a este chip.

```
pacman::p_load("Biobase","GEOquery")
```

Nos bajamos los ficheros CEL.

```
gcel = getGEOSuppFiles("GSE20986")
```

Nos ha creado el subdirectorio *GSE20986* por lo que hemos de cambiar el directorio de trabajo al nuevo directorio.

```
setwd("GSE20986/")
```

Además nos ha bajado un fichero tar del cual hemos de extraer los distintos ficheros CEL. En Linux lo haremos con[5]

```
system("tar xvf GSE20986_RAW.tar")
```

Leemos los datos.

```
gse20986raw = affy::ReadAffy()
```

Añadimos variables fenotípicas.

```
tissue = factor(c(1,2,2,1,2,1,3,3,3,4,4,4),levels = 1:4,
  labels=c("iris","retina","choroides","huvec"))
pd = data.frame(tissue)
rownames(pd) = colnames(gse20986raw)
pData(gse20986raw) = pd
```

Guardamos los datos.

```
save(gse20986raw,file="gse20986raw.rda")
```

En la referencia original el preprocesado de los datos se realizó con el método GC-RMA.[101]. Vamos a reproducirlo.

[101] [128, capítulo 3]

```
pacman::p_load("gcrma","hgu133plus2probe")
gse20986 = gcrma::gcrma(gse20986raw)
```

Y guardamos el nuevo ExpressionSet.[6]

[5] En Windows podemos usar el Winrar.

[6] Los datos que hemos bajado con la función getGEO ya estaban preprocesados utilizando las mismas funciones que usamos aquí. Lo he hecho para mostrar cómo se hace o bien porque podemos querer utilizar algún otro procedimiento de preprocesado.

```
save(gse20986,file="gse20986.rda")
```

En este trabajo se comparan muestras obtenidas a partir de células endoteliales microvasculares de ojos humanos con células endoteliales obtenidas de venas umbilicales humanas (HUVEC). Del primer tipo de células hay tres subtipos dependiendo de dónde se extraen. En concreto se extrajeron del iris, la retina y la coroides. Las células HUVEC son más fáciles de obtener y por lo tanto de estudiar. Sin embargo, no es claro que lo que se estudie con ellas sea extrapolable con los resultados que se obtendrían con los otros tipos de células. En resumen, el objeto fundamental es la comparación de este grupo con los otros tres grupos.

Todo el proceso para su obtención a partir de los datos originales [102] que hemos visto en esta sección lo podemos reproducir lo tenemos en el script http://www.uv.es/ayala/docencia/tami/md/gse20986.html.

[102] Y todo lo que se hace con estos datos en el manual.

5.12 GSE34764

Consideremos los datos de GEO con identificador GSE34764. Necesitamos el paquete [37, GEOquery]. Lo cargamos.

```
library(GEOquery)
```

Obtenemos los ficheros CEL.

```
getGEOSuppFiles("GSE34764")
```

Cambiamos el directorio de trabajo.

```
setwd("GSE30129/")
```

Para leer los datos vamos a utilizar el paquete [30, oligo].

```
library(oligo)
celFiles = list.celfiles()
gse34774 = read.celfiles(celFiles)
save(gse34774,file="gse34774.rda")
```

5.13 ArrayExpress

En § 5.5 hemos visto cómo construir un ExpressionSet cuando tenemos los datos de expresión, los fenotípicos y los de anotación. Si consultamos la referencia original [100] vemos que los autores han subido toda la información del experimento a ArrayExpress. Realmente gran parte del trabajo que hicimos en esa sección la podíamos ahorrar bajando los datos ya preparados de esta base de datos. Lo podemos hacemos hacer con el paquete [74, ArrayExpress] del siguiente modo.

```
library(ArrayExpress)
rawset = ArrayExpress("E-TABM-157")
```

Podemos comprobar que el objeto **rawset** en donde guarda los datos es un **AffyBatch** a la cual podemos aplicar la corrección de fondo, la normalización y el resumen en § 4.

5.14 GSE104645

Esta sección tiene el interés fundamental de que nos ocupamos de un chip Agilent. Estos datos han sido obtenidos utilizando un chip de Agilent. Mostramos el proceso hasta producir un **ExpressionSet**. Los datos son un estudio sobre cáncer colorectal.[103] Podemos encontrar estos datos en https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE104645. Los bajamos con

[103] Siento que todo sea sobre cáncer pero hay muchos bancos de datos.

```
wd = getwd()
setwd(dirTamiData)
GEOquery::getGEOSuppFiles("GSE104645")
setwd("GSE104645")
system("tar xvf GSE104645_RAW.tar")
system("gzip -d *.gz")
x = read.maimages(dir(".", "txt"), "agilent", green.only = TRUE,
                  other.columns="gIsWellAboveBG")
GSE104645raw = x
save(GSE104645raw,file=paste0(dirTamiData,"GSE104645raw.rda"))
setwd(wd)
```

Eliminamos las sondas que no tienen correspondencia ENTREZID.

```
a = AnnotationDbi::select(hgug4112a.db, keys=GSE104645raw$genes[,"ProbeName"],
                          column=c("ENTREZID","ENSEMBL"), keytype="PROBEID")
a = a[!is.na(a[,"ENTREZID"]),]
c1 = match(unique(a[,1]),a[,1])
a1 = a[c1,]
c2 = match(unique(a1[,2]),a1[,2])
a2 = a1[c2,]
a2 = na.omit(a2)
GSE104645raw2 = GSE104645raw[match(a2[,1],GSE104645raw$genes[,"ProbeName"]),]
dim(GSE104645raw2)
rownames(GSE104645raw2) = a2[,1]
save(GSE104645raw2,file=paste0(dirTamiData,"GSE104645raw2.rda"))
```

Hacemos la corrección de fondo.

```
GSE104645el = backgroundCorrect(GSE104645raw2, method="normexp")
```

Ahora normalizamos aplicando una normalización de cuantiles.

```
GSE104645el = normalizeBetweenArrays(GSE104645el, method="quantile")
```

Guardamos los datos.

```
save(GSE104645el,file=paste0(dirTamiData,"GSE104645el.rda"))
```

Loading the previously saved file.

```
load(paste0(dirTamiData,"GSE104645el.rda"))
```

Now we are going to make the `ExpressionSet`.

```
pd0 = data.frame(case = 1:ncol(GSE104645el))
rownames(pd0) = colnames(GSE104645el$E)
metadata = data.frame(labelDescription = c("case"),
                      row.names=colnames(pd0))
phenotypedata = new("AnnotatedDataFrame", data = pd0,
                    varMetadata = metadata)
experimentdata = new('MIAME',name="Okita A, Takahashi S, Ouchi K, Inoue M
et al.",lab="Tohoku University Hospital",contact ="akira.okita.d8@tohoku.ac.jp",
title = "Consensus molecular subtypes classification
 of colorectal cancer as a predictive factor for chemotherapeutic
efficacy against metastatic colorectal cancer",
abstract = "Gene expression profile from 193 formalin-fixed and
paraffin embedded primary tumor samples.",
url = "https://www.ncbi.nlm.nih.gov/pubmed/29721154",
other = list(notes = "An example of Agilent microarray"))

GSE104645 = new("ExpressionSet",exprs=GSE104645el$E,phenoData = phenotypedata,
                experimentData = experimentdata,annotation = "hgug4112a.db")
```

Añadimos el `fData` con los identificadores.

```
fData(GSE104645) = a2[,1:2]
save(GSE104645,file=paste0(dirTamiData,"GSE104645.rda"))
```

5.15 Varios

Esta sección es una especie de cajón de sastre que contiene cosas que no he sabido colocar elegantemente en el resto del tema.

5.15.1 Guardando la matriz de expresión

Supongamos que tenemos un ExpressionSet y queremos guardar la matriz en un fichero externo para leerlo con alguna otra aplicación. Podemos utilizar la función Biobase::write.exprs.

```
data(gse1397,package="tamidata")
```

Guardamos en un fichero texto la matriz de expresión.

```
write.exprs(gse1397,file="gse1397.txt")
```

Y luego podemos abrir este fichero con Calc[104] o con nuestra aplicación preferida.

[104] La versión libre de Excel que tenemos en LibreOffice.

5.16 Ejercicios

Ej. 14 — Se pide construir un ExpressionSet. Este ExpressionSet ha de tener la siguiente matriz de expresión.

```
##        [,1]  [,2]  [,3]  [,4]  [,5]
##  [1,] 24.46 21.59 23.73 22.31 23.73
##  [2,] 23.26 24.41 22.07 23.26 22.65
##  [3,] 23.09 22.50 23.82 22.00 19.78
##  [4,] 20.43 20.78 22.75 22.84 21.51
##  [5,] 24.77 23.58 24.00 23.68 23.69
##  [6,] 21.99 22.43 22.65 24.45 24.36
##  [7,] 23.47 24.02 23.38 23.15 23.83
##  [8,] 24.06 24.11 21.36 21.86 23.34
##  [9,] 24.44 22.62 23.35 22.20 20.60
## [10,] 24.58 21.44 24.66 23.41 24.38
## [11,] 22.77 25.95 23.20 21.70 24.72
## [12,] 23.66 23.93 24.28 23.25 22.42
## [13,] 20.49 23.83 21.10 21.44 25.75
## [14,] 22.97 20.95 22.54 22.56 19.35
## [15,] 24.07 24.33 24.04 22.23 21.75
## [16,] 22.60 22.62 23.67 24.31 23.30
## [17,] 21.76 22.58 21.27 23.40 23.08
## [18,] 23.52 24.25 21.79 22.62 21.22
## [19,] 23.25 22.39 21.87 22.79 24.96
## [20,] 21.14 24.22 23.06 23.06 22.19
```

Los nombres de las filas ha de ser

```
##  [1] "g1"  "g2"  "g3"  "g4"  "g5"  "g6"  "g7"
##  [8] "g8"  "g9"  "g10" "g11" "g12" "g13" "g14"
## [15] "g15" "g16" "g17" "g18" "g19" "g20"
```

Los nombres de las muestras serán

```
##  [1] "m1"  "m2"  "m3"  "m4"  "m5"  "m6"  "m7"
##  [8] "m8"  "m9"  "m10" "m11" "m12" "m13" "m14"
## [15] "m15" "m16" "m17" "m18" "m19" "m20"
```

Como datos fenotípicos (covariables que describen las columnas) han de ser las siguientes

```
##   tipo crecimiento
## 1    1        0.30
## 2    2        0.23
## 3    2        0.34
## 4    1        0.24
## 5    2        0.45
```

Ej. 15 — [7] Vamos a bajar y preprocesar los datos *GSE30129* de GEO. Se pide realizar los siguientes pasos.

1. Bajar los datos a nivel de sonda utilizando el paquete [37, GEOquery] y la función `getGEOSuppFiles`.
2. Leer los datos utilizando la función `read.celfiles` del paquete [30, oligo].
3. En el paso anterior si no tenemos instalado el paquete `pd.mogene.1.0.st.v1` nos dará un aviso.

[7] Este problema es muy similar a lo que hacemos en § 5.12.

4. Instalar el paquete indicado y repetir la lectura.
5. Representar los estimadores de densidades y los diagramas de cajas de las expresiones a nivel de sonda.
6. Obtener los MAplots.
7. Aplicar el método RMA.
8. Aplicar el método MAS5. ¿Es posible hacerlo?
9. Repetir los dibujos de los apartados 5 y 6 para los datos procesados utilizando el método RMA.

* **Ej. 16** — Consideremos los datos tamidata::gse20986raw.

1. ¿Cómo identifica los genes de las filas?
2. Cambiar los identificadores de los genes a sus códigos Ensembl.

Capítulo 6

Datos de RNA-seq

6.1 Introducción

En este tema trabajamos con datos obtenidos mediante la técnica conocida como RNA-Seq. En `http://rnaseq.uoregon.edu/` tenemos una introducción muy simple y clara. [103] da una buena visión general. En [**R-RnaSeqTutorial**] tenemos una presentación general de cómo trabajar con estos datos en el contexto de R/Bioconductor. Un texto general que trata lo relativo al análisis de este tipo de datos es [77]. Una referencia más breve pero muy interesante es [35] donde también da una visión global del análisis de este tipo de información.

En este manual se asume que se trabaja con un genoma de referencia. Los procedimientos que vamos a estudiar asumen este punto de partida.

De un modo análogo a § 4 y § 5 comentaremos este tipo de información de expresión génica.[105]

[105] En el momento actual uno de los que más interés despierta.

6.2 Flujo de trabajo con RNA-seq

Un estudio de este tipo implica la realización de los siguientes pasos ([103]):

1. Diseño experimental.
2. Protocolos de extracción del RNA.
3. Preparación de las librerías. Se convierte el RNA en cDNA y se añaden los adaptadores para la secuenciación.
4. Se secuencian las lecturas cDNA utilizando una plataforma de secuenciación.
5. Alineamiento de las lecturas secuenciadas a un genoma de referencia.
6. Resumen del número de lecturas alineadas a una región.
7. Normalización de las muestras para eliminar diferencias técnicas en la preparación.
8. Estudio estadístico de la expresión diferencial incluyendo en lo posible un modelo.

9. Interpretación de los resultados desde el punto de vista biológico.

En este manual estamos interesados fundamentalmente en el análisis de expresión diferencial. Obviamente el punto 1 es básico. Lo realiza el experimentador. En la medida en que vamos a trabajar con datos de experimentos ya realizados hablaremos de cuestiones de diseño aunque no de un modo central. La extracción del RNA, preparación de librerías y la secuenciación son cuestiones no tratadas aquí.

Sí que comentaremos cómo alinear las lecturas cuando disponemos de un genoma de referencia y de los pasos que siguen en el análisis. Sin embargo, este manual se quiere centrar en problemas estadísticos en el contexto de la Bioinformática. Por ello, toda la parte de alineamiento y conteo de las lecturas lo veremos de un modo sencillo insistiendo en cómo hacerlo y no en qué se hace. Nuestro trabajo empieza a partir del momento en que tenemos los conteos y nos interesamos por estudiar si estos conteos son significativamente distintos entre distintas condiciones biológicas.[106]

[106] En [6] tenemos el análisis detallado con todos los pasos desde las lecturas originales hasta el análisis de expresión diferencial final. El interés del trabajo es que podemos reproducir un análisis completo. Además contiene enlaces para aquellas herramientas que se utilizan fuera de R/Bioconductor como bowtie2, samtools y tophat2. En `http://www.bioconductor.org/help/workflows/rnaseqGene/` tenemos todo un flujo en donde se analiza un experimento. El paquete [**R-airway**] tiene los datos para poder trabajar.

En el diseño hemos de diferenciar entre **réplicas técnicas** en la que utilizamos una misma muestra biológica, un mismo procesado y los mismos protocolos de secuenciación. Es de esperar que las diferencias de lo que observamos en estas réplicas técnicas sean menores a las obtenidas con **réplicas biológicas** en las que se utilizan distintas muestras biológicas. En los experimentos RNA-seq solemos tener réplicas biológicas y raramente réplicas técnicas.

¿Cuántas muestras? Se trabaja con muestras extremadamente pequeñas: 2 o 3 por tratamiento. Obviamente **no** podemos esperar ver diferencias claras entre tratamientos.

Un diseño experimental siempre ha de ser simple. Este es un comentario aplicable a cualquier tipo de dato pero es particularmente importante con dato ómico. Sin duda, en general y en particular en este contexto, los diseños experimentales han de mantenerse lo más simples que se pueda.

¿Qué tipos de tratamiento hemos de realizar? Es necesaria una normalización previa de las muestras que permita "controlar" las variaciones técnicas.

1. Por ejemplo, las muestras proporcionadas por el secuenciador tienen distintas cantidades de DNA y esto da lugar a diferencias en el número total de lecturas secuenciadas y alineadas independientemente de las diferente expresión diferencial de un gen. El tratamiento debiera afectar solamente a una fracción de los genes evaluados. Si la diferencia se observa en "todos" los genes lo que vemos es un efecto del protocolo de trabajo.

2. También hay diferencias entre genes que no se deben a diferencias de expresión. En los protocolos estandar un gen largo es secuenciado con más frecuencia que un gen corto.

En las secciones que siguen se comentan los datos que utilizamos en los restantes capítulos.

La clase que en R/Bioconductor debe de utilizarse para trabajar con datos RNA-seq es GenomicRanges::SummarizedExperiment.

Responder las siguientes preguntas es el objeto de este tema.

1. ¿Dónde podemos obtener datos de secuenciación? De otro modo: ¿qué repositorios podemos utilizar?

2. ¿Cómo podemos realizar búsquedas en los metadatos con objeto de obtener experimentos que tenga que ver con lo que me interesa?

3. ¿Cómo bajarlos?

4. ¿Cómo contar las lecturas alineadas sobre intervalos o uniones de intervalos sobre el genoma?

6.3 Repositorios

Los repositorios donde podemos encontrar este tipo de datos son NCBI en los Estados Unidos, EMBL en Europa y DDBJ en Japón.[107] En NCBI GEO también hay datos de secuencias y podemos acceder a ellos utilizando [37, GEOquery]. En concreto las direcciones son

[107] Son los básicos como para datos de microarrays tenemos NCBI GEO y EBI ArrayExpress.

NCBI SRA `http://www.ncbi.nlm.nih.gov/sra` Posiblemente es la mayor de las bases de datos. No almacena lecturas alineadas. En NCBI GEO sí que tenemos algunos bancos de datos con lecturas alineadas sobre un genoma de referencia.

EBI ENA `http://www.ebi.ac.uk/ena`

DDBJ `http://www.ddbj.nig.ac.jp`

En lo que sigue utilizamos fundamentalmente la primera base de datos y, a veces, la segunda. Cada una de estas bases de datos puede ser consultada en línea y bajar los datos desde la propia página. También usaremos en lo que sigue herramientas para hacerlo desde R, por ejemplo, [156, SRAdb].

En las secciones § 6.4 y § 6.5. comentamos distintos formatos para almacenar lecturas cortas. También comentamos paquetes R/Bioconductor para trabajar con estos formatos.

6.4 Formato FASTA

Es el formato basado en texto para representar secuencias bien de nucleótidos bien de péptidos. Tanto unos como otros son representados por una sola letra. También tiene símbolos para representar un hueco (gap) o parada en la traducción o bien que no se sabe el nucleótido o aminoácido. Es muy simple. Tiene una línea que tiene en la primera posición el símbolo > al que sigue una descripción de la secuencia. En la siguiente línea empieza la secuencia de bases o aminoácidos. Se recomiento que no tener más de 80 columnas y se pueden tener todas las filas que se precisen.

6.5 Formato FASTQ

Es uno de los formatos más populares para datos de secuencias. Consiste de cuatro líneas. La primera contiene el nombre de la secuencia. La segunda línea contiene a la propia secuencia. La tercera línea contiene información opcional sobre la secuencia. La cuarta línea cuantifica la confianza o calidad en la determinación de cada base recogida en la segunda línea. Las cuatro siguientes líneas corresponden a un ejemplo.

```
@SRR1293399.1 ILLUMINA-545855_0026_FC629BG:6:1:1022:5049 length=50
ACAGGGACGCCATCGAATCCGGATCNTNNNNNNNNNNNNANNNNNNNNNN
+SRR1293399.1 ILLUMINA-545855_0026_FC629BG:6:1:1022:5049 length=50
dee\edYcdc`bbY`S]bb_]]Ua^BBBBBBBBBBBBBBBBBBBBBBBBB
```

¿Cómo se cuantifica la confianza o precisión? Se utiliza el programa Phred.[108] Este programa lo que hace es asignar los picos de fluorescencia a una de las cuatro bases (o *base call*). En [**EwingGreen98**] tenemos la explicación del método de asignación. Si P denota la probabilidad[109] para una base dada de ser mal asignada o clasificada entonces el valor con el que se trabaja es

$$Q = -10 \log_{10} P. \tag{6.1}$$

Esto significa que una probabilidad P muy pequeña de clasificación incorrecta se traduce en un valor grande de Q. Una vez calculado Q estonces se asigna (usualmente) el caracter ASCII que ocupa la posición $33+Q$.[110] Una descripción detallada sobre FASTQ la tenemos en [34].

[108] El procedimiento que utiliza aparece en [**Ewing98**] y cómo estima las probabilidades de error cuando asigna cada base aparece en [**EwingGreen98**].

[109] No es realmente una probabilidad. Es una cuantificación de la calidad de la asignación y no más.

[110] Sin duda, un método curioso de mostrar valores numéricos utilizando texto.

Utilizando ShortRead

Para trabajar con el formato FASTQ es útil el paquete [95, ShortRead].

```
library(ShortRead)
```

Vamos a ver sus funcionalidades utilizando `SRR1293399_1.fastq`. En esta sección suponemos fijado el directorio de trabajo a donde tenemos los datos.

Leemos con ShortRead::readFastq.[111]

[111] Veremos que tarda lo suyo en leer los datos.

```
fq0 = readFastq("SRR1293399_1.fastq")
```

Es un fichero con muchas lecturas y para trabajar es mejor tomar una muestra de las mismas. Tomamos 10000 lecturas elegidas al azar. Y lo hacemos con ShortRead::FastqSampler. Volvemos a leer esta muestra aleatoria.

```
fqsample = FastqSampler("SRR1293399_1.fastq",10000)
fq0 = yield(fqsample)
```

Ahora `fq0` es un ejemplo de clase

```
class(fq0)

## [1] "ShortReadQ"
## attr(,"package")
## [1] "ShortRead"
```

En particular podemos ver la información básica con

```
fq0

## class: ShortReadQ
## length: 10000 reads; width: 50 cycles
```

[112] Subsetting

Obviamente tiene solamente 10000 lecturas. Podemos seleccionar[112] del modo habitual. Por ejemplo, la primera sería

```
fq0[1]

## class: ShortReadQ
## length: 1 reads; width: 50 cycles
```

O de la tercera a la décima con

```
fq0[3:10]

## class: ShortReadQ
## length: 8 reads; width: 50 cycles
```

Con ShortRead::sread podemos tener la lectura. Por ejemplo, la que ocupa la posición 1000 sería

```
sread(fq0[1000])

## DNAStringSet object of length 1:
##     width seq
## [1]    50 AGAGAAATCGGTGTCAGTC...GTATGCCGTCTTCTGCTT
```

Podemos ver las primeras y últimas con

```
sread(fq0)

## DNAStringSet object of length 10000:
##         width seq
##     [1]    50 TAAACACTGTCTCCTTG...CCTGGTGTCAGTCACT
##     [2]    50 ACAGCTTTGTCTGAGAC...TGTCAGTCATTCCAGC
##     [3]    50 GTGTCAGTCACTTCCAG...GGTCGTATGCCGTCTT
##     [4]    50 TGCGGCCCCGGGTTCCT...CTGAGCGTGTCAGTCA
##     [5]    50 TCTTTGGGTTCCGGGGG...TGAAACGTGTCAGTCA
##     ...   ... ...
##  [9996]    50 ATAGTCTGTGGGAGTCA...CACTTCCAGCGGTCGT
##  [9997]    50 CAAAGTGCTTACAGTGC...CCAGCGGTCGTATGCC
##  [9998]    50 GTGTCAGTCACTTCCAG...CTGCTTGAAAAAAAAA
##  [9999]    50 TGGCGCTGCGGGATGAA...GCGCCCGATGCCGACG
## [10000]    50 CAACTAGCCCTGAAAAT...CAGGCCCATACCCGTG
```

Podemos ver la longitud de las lecturas con

```
width(fq0)
```

Con ShortRead::detail tenemos una información detallada. Las medidas de calidad de la lectura 1000 las obtenemos con

```
quality(fq0[1000])

## class: SFastqQuality
## quality:
## BStringSet object of length 1:
##     width seq
## [1]    50 ffffffffffffffffeff...fffeffefffff`ddbdd
```

Podemos saber el valor a que corresponde la codificación de la calidad con

```
encoding(quality(fq0))
```

```
##  ;  <  =  >  ?  @  A  B  C  D  E  F  G  H  I  J
## -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8  9 10
##  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z
## 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
##  [ \\  ]  ^  _  `  a  b  c  d  e  f  g  h  i
## 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
```

```
fls = dir(paste(dirTamiData,"SRP042140/fastq",sep=""),"*fastq$",full=TRUE)
```

Evaluando la calidad del experimento. Vamos a realizar un control de calidad del experimento con ShortRead::qa. Lo hacemos para la primera muestra.[113]

[113] Obviamente debemos repetir el análisis para cada una de las muestras.

```
SRP042140_1.qa = qa(fls[1],type="fastq")
```

Generamos un informe y lo visualizamos.[114]

[114] A los usuarios de Windows es probable que el codigo que sigue no les funcione. ¿Les extraña?

```
browseURL(report(SRP042140_1.qa))
```

Es un informe muy interesante. Notemos que estos datos ya están previamente filtrados[115] y por ello algunas de las características evaluadas no tienen sentido.

[115] Son datos obtenidos de un repositorio público.

Podemos ver la información contenida en `SRP042140.qa` con

```
show(SRP042140_1.qa)
```

```
## class: FastqQA(10)
## QA elements (access with qa[["elt"]]):
##   readCounts: data.frame(1 3)
##   baseCalls: data.frame(1 5)
##   readQualityScore: data.frame(512 4)
##   baseQuality: data.frame(94 3)
##   alignQuality: data.frame(1 3)
##   frequentSequences: data.frame(50 4)
##   sequenceDistribution: data.frame(357 4)
##   perCycle: list(2)
##     baseCall: data.frame(249 4)
##     quality: data.frame(1800 5)
##   perTile: list(2)
##     readCounts: data.frame(0 4)
##     medianReadQualityScore: data.frame(0 4)
##   adapterContamination: data.frame(1 1)
```

Por ejemplo podemos ver el número de lecturas.

```
SRP042140_1.qa[["readCounts"]]

##                         read filter aligned
## SRR1293399_1.fastq 29508968     NA      NA
```

O también podemos ver las frecuencias de cada una de las bases.

```
SRP042140_1.qa[["baseCalls"]]

##                              A        C        G
## SRR1293399_1.fastq 10600637 12762822 12873094
##                              T    N
## SRR1293399_1.fastq 13753938 9509
```

O las lecturas más frecuentes.

```
head(SRP042140_1.qa[["frequentSequences"]])

##                                                sequence
## 1 GTGTCAGTCACTTCCAGCGGTCGTATGCCGTCTTCTGCTTGAAAAAAAAA
## 2 TAGCTTATCAGACTGATGTTGACGTGTCAGTCACTTCCAGCGGTCGTATG
## 3 TGTAAACATCCCCGACTGGAAGCGTGTCAGTCACTTCCAGCGGTCGTATG
## 4 AACTGGCCCTCAAAGTCCCGCTGTGTCAGTCACTTCCAGCGGTCGTATGC
## 5 TGTCAGTCACTTCCAGCGGTCGTATGCCGTCTTCTGCTTGAAAAAAAAAA
## 6 TGTAAACATCCCCGACTGGAAGCTGTGTCAGTCACTTCCAGCGGTCGTAT
##   count type               lane
## 1 94635 read SRR1293399_1.fastq
## 2 55085 read SRR1293399_1.fastq
## 3 37265 read SRR1293399_1.fastq
## 4 28687 read SRR1293399_1.fastq
## 5 18234 read SRR1293399_1.fastq
## 6 13152 read SRR1293399_1.fastq
```

Podemos ver la frecuencia de cada base en cada posición de la lectura. Por ejemplo, para las dos primeras posiciones.

```
head(SRP042140_1.qa[["perCycle"]]$baseCall,n=10)

##    Cycle Base  Count               lane
## 1      1    A 221622 SRR1293399_1.fastq
## 2      1    C  70578 SRR1293399_1.fastq
## 3      1    G 202760 SRR1293399_1.fastq
## 4      1    T 504717 SRR1293399_1.fastq
## 15     1    N    323 SRR1293399_1.fastq
## 19     2    A 350254 SRR1293399_1.fastq
## 20     2    C  97623 SRR1293399_1.fastq
## 21     2    G 280410 SRR1293399_1.fastq
## 22     2    T 271675 SRR1293399_1.fastq
## 33     2    N     38 SRR1293399_1.fastq
```

```
head(SRP042140_1.qa[["perCycle"]]$quality,n=10)

##    Cycle Quality Score Count               lane
## 35     1       B     2   678 SRR1293399_1.fastq
```

```
## 44     1       K    11    80 SRR1293399_1.fastq
## 45     1       L    12  3411 SRR1293399_1.fastq
## 46     1       M    13   625 SRR1293399_1.fastq
## 51     1       R    18   137 SRR1293399_1.fastq
## 53     1       T    20  6936 SRR1293399_1.fastq
## 54     1       U    21   907 SRR1293399_1.fastq
## 55     1       V    22  1140 SRR1293399_1.fastq
## 56     1       W    23    88 SRR1293399_1.fastq
## 57     1       X    24   757 SRR1293399_1.fastq
```

6.6 De SRA a fastq

Lo hacemos con SRA Toolkit.[116] Utilizamos la función `fastq-dump`.

```
fastq-dump -I --split-files nombre_fichero.sra
```

Y lo repetimos para cada fichero SRA.

Una opción que nos lo hace para todos es[117]

```
cat files | parallel -j 7 fastq-dump --split-files {}.sra
```

donde `files` es un fichero que tiene en cada línea el nombre del fichero sra sin la extensión. En nuestro caso el fichero `files` es

```
SRR1293399
SRR1293400
SRR1293401
SRR1293402
SRR1293403
SRR1293404
SRR1293405
SRR1293406
SRR1293407
```

[116] En Debian/Ubuntu hemos de instalar el paquete sra-toolkit. Consultar https://github.com/ncbi/sra-tools/wiki/Downloads.

[117] Previamente hay que instalar *parallel* con *apt-get install parallel*.

6.7 Control de calidad de un experimento RNA-seq con EDASeq

El control de calidad se puede realizar en dos pasos. En el primero se trata de evaluar si tenemos librerías (muestras) con bajas profundidades, o con problemas de calidad o con frecuencias de nucleóticos no frecuentes. En un segundo paso, a nivel de gen, buscamos etiquetas equivocadas asociadas a las muestras, problemas con las distribuciones de probabilidad supuesta (sobredispersión es el gran problema) y sesgos en los contenidos GC.

EDA al nivel de lectura. En particular, tenemos que considerar los siguientes aspectos:

Número de lecturas alieanadas y no alineadas. Si hay un total de pocas lecturas esto se atribuye a la mala calidad del RNA. Si hay una tasa de alineamiento bajo es que las lecturas tienen poca calidad, normalmente tienen secuencias poco complejas de difícil alineamiento. También podemos tener un mal genoma de referencia o simplemente mal etiquetadas las muestras.

6.8 STAR y samtools

Veamos cómo alinear y contar utilizando conjuntamente STAR y samtools.

1. Instalamos STAR.
2. Copiamos la versión Linux en el directorio donde tenemos los ficheros SRA.
3. Creamos dos subdirectorios con los nombres `aligned` y `fastq`.
4. Nos bajamos (con `wget`) de http://labshare.cshl.edu/shares/gingeraslab/www-data/dobin/STAR/STARgenomes/ENSEMBL/homo_sapiens la última versión.[1]
5. Descomprimimos el fichero anterior y extraemos los ficheros del archivo tar en el subdirectorio `ENSEMBL.homo\_sapiens.release-75`.
6. Supongamos que:
   (a) El ejecutable `STAR` lo tenemos en nuestro directorio de trabajo.
   (b) `ENSEMBL.homo\_sapiens.release-75` es un subdirectorio de nuestro directorio de trabajo.
   (c) Los ficheros FASTQ los tenemos en el subdirectorio `fastq`.
   (d) Pretendemos guardar el fichero SAM en el subdirectorio `aligned`.

   Entonces el código (utilizando STAR) para generar el fichero SAM a partir del FASTQ sería el siguiente

```
./STAR --genomeDir ENSEMBL.homo_sapiens.release-75
    ↪ --readFilesIn
fastq/SRR1293399_1.fastq --runThreadN 12 --
    ↪ outFileNamePrefix aligned/SRR1293399_1.
```

   Podemos comprobar que el fichero generado es `SRR1293399_1.Aligned.out.sam`. Y el código anterior lo hemos de repetir para cada uno de nuestros ficheros FASTQ.

De SAM a BAM utilizando Samtools.

Utilizamos Samtools para transformar los ficheros SAM en ficheros BAM.

```
samtools sort view -bS aligned/SRR1293399_1.Aligned.out.
    ↪ sam -o aligned/SRR1293399_1.bam
```

Contando las lecturas con Rsamtools.

1. Suponemos que los ficheros `bam` generados previamente están en el subdirectorio `aligned` de nuestro directorio de trabajo.
2. Creamos el fichero `bamfiles.txt` que tenga en cada línea el nombre de uno de los ficheros BAM. Por ejemplo, las tres primeras líneas de este fichero podrían ser

[1] En el momento de escribir *ENSEMBL.homo_sapiens.GRCh38.release-79*.

```
SRR1293399_1.bam
SRR1293400_1.bam
SRR1293401_1.bam
```

3. Creamos la lista de ficheros bam.

```
library(Rsamtools)
dirActualData = paste(dirTamiData,"SRP042140_sra/",sep="")
sampleTable = read.table(paste(dirActualData,"bamfiles.txt",sep=""))
fls = paste(dirActualData,"aligned/",sampleTable[,1],sep="")
bamLst = BamFileList(fls, index=character(),yieldSize=100000,obeyQname=TRUE)
```

4. Contamos las lecturas.

   (a) Cargamos [23, GenomicFeatures].

   ```
   library(GenomicFeatures)
   ```

   (b) Leemos el fichero GTF/GFF que hemos bajado previamente.

   ```
   gtffile = paste(dirTamiData,
       "ENSEMBL.homo_sapiens.release-75/Homo_sapiens.GRCh37.75.gtf",sep="")
   ```

   (c)
   ```
   txdb = makeTxDbFromGFF(gtffile, format="gtf")
   genes = exonsBy(txdb, by="gene")
   library(GenomicAlignments)
   SRP042140.counts = summarizeOverlaps(features = genes, read=bamLst,
       mode="Union",
       singleEnd=TRUE, ## No son lecturas apareadas
       ignore.strand=TRUE,
       fragments=FALSE)
   save(SRP042140.counts,
        file=paste(dirTamiData,"SRP042140_sra/","SRP042140.counts.rda"),sep="")
   ```

Figura 6.1: Un anuncio desactualizado según la OMS.

6.9 Bowtie2

[118]

Es una herramienta software para alinear lecturas cortas sobre secuencias de referencia largas. Soporta distintos modos de alineamiento: con huecos[119], locales y con lecturas apareadas[120]. La salida del programa la realiza en formato SAM.

El programa `bowtie2` utiliza un fichero de índices de clase Bowtie 2 y ficheros con lecturas secuenciadas y produce un fichero en formato SAM.

¿Qué es alinear? Tomamos una lectura o secuencia corta y pretendemos encontrar en una secuencia larga donde es **más** similar, en que punto hay una mayor similitud respecto de la secuencia larga o de referencia. ¿Cuál es el resultado que obtenemos después de alinear? La secuencia corta es colocada sobre una parte de la secuencia de referencia indicando en qué puntos se produce una correspondencia

[118] http://bowtie-bio.sourceforge.net/bowtie2/manual.shtml

[119] gapped

[120] Paired-end

e indicando los huecos[121] que se han tenido que introducir en una u otra de las secuencias para conseguir la mejor correspondencia posible entre dichas secuencias.

[121] Gaps.

```
Read:      GACTGGGCGATCTCGACTTCG
           |||||  |||||||||| |||
Reference: GACTG--CGATCTCGACATCG
```

Mediante la línea vertical mostramos en qué puntos hay una correspondencia correcta mientras que con un guión en una u otra secuencia indicamos en qué posiciones hemos debido de incorporar un hueco para poder hacer corresponder ambas secuencias.

Notemos que no estamos seguros de que efectivamente la secuencia corta se haya originado en este punto. Es una hipótesis. La mejor que se puede conjeturar pero no es seguro ni mucho menos. En muchas ocasiones no se puede encontrar una correspondencia.

Dos tipos de alineamientos son considerados: **alineamiento de extremo a extremo** o alineamiento global[122] y alineamiento local.[123] En la primera opción toda la secuencia corta es alineada. En el segundo tipo permitimos que uno o los dos extremos contengan bases no alineadas. En Bowtie2 la opción local se indica `--local` para el alineamiento local mientras que la opción global se asume por defecto.

[122] End-to-end alignment.

[123] Local alignment.

Hemos de evaluar la calidad del alineamiento. Para ello definimos una puntuación asociada a cada posible correspondencia. Lo que podemos llamar una función objetivo. La idea es que cuanto mayor sea la función objetivo mejor es el alineamiento. Realmente penalizamos cada error en la correspondencia. Tendremos una penalización por correspondencia errónea, una penalización por cada hueco. En el caso de alineamiento local sumaremos un valor por cada correspondencia. Finalmente el valor de la función objetivo nos cuantifica la calidad final de la correspondencia conseguida. En concreto se consideran los siguientes valores que se suman (bajando o subiendo) para calcular el valor de la función objetivo.

–**ma** Sumamos un valor positivo por cada correspondencia (para alineamiento local).

–**mp** Penalización (restamos) por una correspondencia errónea.

–**np** Penalización por tener un valor N (esto es una base no especificada) en la secuencia corta o en la larga de referencia.

–**rdg** Penalización por hueco en secuencia corta.

–**rfg** Penalización por hueco en secuencia de referencia.

Obviamente mediante algún procedimiento encontraremos siempre una correspondencia. No necesariamente esta correspondencia la hemos de considerar suficientemente buena. ¿Cuándo la consideraremos aceptable? Lo que se hace es fija un umbral por debajo del cual el alineamiento no se acepta. Este umbral lógicamente ha de depender de la longitud de la lectura corta. El valor de este umbral que por defecto (aunque es configurable) tiene definido Bowtie2 es, para alineamiento global,

$$-0.6 - 0.6 \cdot L, \tag{6.2}$$

siendo L la longitud de la lectura. Para alineamiento local el valor es

$$20 + 8 \cdot ln(L). \tag{6.3}$$

Lo podemos configurar con la opción `--score-min`.

¿Podemos alinear de un modo único? Por supuesto que no. Pensemos en zonas de la secuencia en donde se repiten mucho los elementos. Si tenemos secuencias cortas entonces podremos alinear (perfectamente) de distintos modos. Tendremos distintos alineamientos igualmente buenos. Se plantea el problema de elegir entre estos alineamientos. Esto se puede hacer utilizando las medidas de calidad de la correspondencia.

[124] Paired-end o mate-pair. No es lo mismo.

¿Qué son lecturas apareadas?[124] Son pares de lecturas cortas o secuencias cortas de las cuales conocemos a priori su posición relativa y la distancia que las separa en la lectura de DNA. Cuando tenemos lecturas apareadas tendremos dos ficheros que incluyen el primer y el segundo elemento del par, uno en cada fichero. En concreto la misma línea en cada fichero indica cada una de las componentes del par de lecturas. En Bowtie2 se utilizan los argumento `-1` y `-2` para indicar el primer y segundo fichero. Cuando se realiza el alineamiento de un par de secuencias cortas en el fichero SAM resultante se indica en los campos `RNEXT` y `PNEXT` el nombre y la posición del otro elemento del par. También se indica la longitud del fragmento de DNA del cual se secuenciaron los dos fragmentos. Cuando se realiza el alineamiento de un par de lecturas este se puede producir de un modo concordante o de un modo discordante. En un modo concordante los dos elementos del par se alinean verificando lo que se espera en orientación y distancia entre ellos. El alineamiento es discordante entonces tenemos para cada uno **un alineamiento único** pero no se verifican las restricciones de orientación y distancia. Por defecto, Bowtie2 busca tanto alineamientos concordantes como alineamientos discordantes. Con la opción `--no-discordant` solamente busca concordantes.

En la opción por defecto, Bowtie2 realiza un alineamiento mixto. En esta opción primero busca alinear el par de lecturas de un modo concordante. Si no lo logra entonces busca alinear cada una de ellas de un modo discordante. Si no queremos la segunda parte de la búsqueda podemos eliminarla con la opción `--no-mixed`.

Bowtie2 busca alineamientos válidos para cada lectura. Cuando encuentra un alineamiento válido sigue buscando otros que sean al menos tan buenos como el que ha encontrado. Tiene unos controles para dejar de buscar. Cuando cesa la búsqueda utiliza los mejores alineamientos que ha encontrado para evaluar la calidad de la correspondencia encontrada (el campo `MAPQ` en el formato SAM).

En `http://bowtie-bio.sourceforge.net/bowtie2/manual.shtml` se tiene una muy buena exposición de todas las opciones que ofrece este programa y es más que recomendable su lectura.

Ejemplo 6.1 *Vamos a realizar el alineamiento utilizando Bowtie2.*

1. *Los datos corresponden a Illumina.*
2. *Bajamos el fichero de índices de* `http://support.illumina.com/sequencing/sequencing_software/igenome.html`.
3. *En concreto bajamos* `\Homo_sapiens_Ensembl_GRCh37.tar.gz`.
4. *Ejecutamos en línea de comandos.*

```
bowtie2 -x ~/DOCENCIA/tami-data/Homo_sapiens/Ensembl
    ↪ /GRCh37/Sequence/Bowtie2Index/genome -U fastq
    ↪ /SRR1293399_1.fastq -S aligned/SRR1293399_1.
    ↪ sam
```

6.10 Utilizando tophat y samtools

Seguimos los siguientes pasos.

1. Necesitamos el genoma humano de referencia. Vamos a utilizar GRCh37.
2. Lo podemos encontrar en http://ccb.jhu.edu/software/tophat/igenomes.shtml. Lo bajamos.
3. Descomprimimos el fichero anterior.

```
gzip -d Homo_sapiens_Ensembl_GRCh37.tar.gz
```

4. Alineamos con

```
tophat2 -o file_tophat_out genome SRR479053_1.fastq
    ↪ SRR479053_2.fastq
samtools sort -n file_tophat_out/accepted_hits.
    ↪ bam_sorted
```

Antes de realizar el alineamiento necesitamos el genoma humano de referencia. Se utiliza GRCh37. En concreto vamos a utilizar el índice Bowtie que podemos encontrar en http://ccb.jhu.edu/software/tophat/igenomes.shtml.

Descomprimimos el fichero anterior.

```
gzip -d Homo_sapiens_Ensembl_GRCh37.tar.gz
```

Alineamos.

```
tophat2 -o file_tophat_out genome SRR479053_1.fastq
    ↪ SRR479053_2.fastq
samtools sort -n file_tophat_out/accepted_hits.bam_sorted
```

6.11 GSE37211: paquete parathyroidSE

Estos datos corresponden al experimento con número de acceso GEO GSE37211. En la viñeta del paquete [82, parathyroidSE] tenemos una descripción detallada para obtener los conteos a nivel de gen o de exon partiendo de los ficheros originales sra.[125] Ya los tenemos en el paquete [82, parathyroidSE].

[125] Primero hay que bajarlos pacientemente.

```
data(parathyroidGenesSE,package="parathyroidSE")
se = parathyroidGenesSE
```

Son datos relativos a los genes. ¿Qué clase tenemos?

```
class(se)

## [1] "RangedSummarizedExperiment"
## attr(,"package")
## [1] "SummarizedExperiment"
```

Es un SummarizedExperiment::RangedSummarizedExperiment y será nuestra clase de referencia cuando trabajamos con datos de RNA-seq.[126] ¿Cuántos genes y muestras tenemos?

[126] Es muy conveniente leer la viñeta de [97, SummarizedExperiment].

```
dim(se)

## [1] 63193    27
```

La matriz con los conteos nos la da GenomicRanges::assay.[127]

[127] Es la análoga a Biobase::exprs para Biobase::ExpressionSet.

```
assay(se)
```

Veamos las tres primeras filas (genes) y las tres primeras columnas (muestras).

```
assay(se)[1:3,1:3]

##                 [,1] [,2] [,3]
## ENSG00000000003  792 1064  444
## ENSG00000000005    4    1    2
## ENSG00000000419  294  282  164
```

Tenemos los metadatos de las columnas o variables fenotípicas o covariables asociadas a las muestras con GenomicRanges::coldata.[128]

[128] Corresponde con Biobase::pData.

```
colData(se)
```

Es un IRanges::DataFrame.

```
class(colData(se))

## [1] "DataFrame"
## attr(,"package")
## [1] "IRanges"
```

Por ejemplo, podemos ver los nombres de las variables fenotípicas con

```
names(colData(se))

## [1] "run"        "experiment" "patient"
## [4] "treatment"  "time"       "submission"
## [7] "study"      "sample"
```

Y acceder a los valores de la variable `treatment` con

```
colData(se)[,"treatment"]
```

```
##  [1] Control Control DPN     DPN     OHT
##  [6] OHT     Control Control DPN     DPN
## [11] DPN     OHT     OHT     OHT     Control
## [16] Control DPN     DPN     OHT     OHT
## [21] Control DPN     DPN     DPN     OHT
## [26] OHT     OHT
## Levels: Control DPN OHT
```

o bien con (no mostrado)

```
colData(se)$treatment
```

La información sobre las filas que, en este caso, corresponde con genes.

```
rowRanges(se)
```

Por tanto, las filas de un SummarizedExperiment es un GenomicRanges::GRangesList de modo que cada fila es un GenomicRanges::GRanges indicando los exones que se utilizaron para contar las secuencias de RNA. Los nombres de los genes los podemos obtener con

```
rownames(se)
```

Los primeros con

```
head(rownames(se))
```

```
## [1] "ENSG00000000003" "ENSG00000000005"
## [3] "ENSG00000000419" "ENSG00000000457"
## [5] "ENSG00000000460" "ENSG00000000938"
```

[129]

[129] En esta sección hemos visto cómo manejar un la clase GenomicRanges::SummarizedExperiment y sus métodos asociados.

6.12 GSE64099

Bajamos los datos de GEO.

```
wget ftp.ncbi.nlm.nih.gov/geo/series/GSE64nnn/GSE64099/
    ↪ suppl/GSE64099_RAW.tar
tar xvf GSE64099_RAW.tar
gzip -d *
rm GSE64099_RAW.tar
```

Se tienen las siguientes muestras:

```
GSM1564328_2086.txt  GSM1564331_241.txt   GSM1564329_2433.txt
GSM1564332_1932.txt  GSM1564327_1934.txt  GSM1564330_3225.txt
GSM1564333_1940.txt
```

Construimos un `ExpressionSet`.

```
library(Biobase)
exprs0 = CountAll
rownames(exprs0) = EntrezAll
colnames(exprs0) = c("GSM1564328","GSM1564331","GSM1564329","GSM1564332",
            "GSM1564327","GSM1564330","GSM1564333")
type = c(2,1,2,1,1,2,2)
type = factor(type,levels=1:2,labels=c("Wild-type","Smchd1"))
type = data.frame(type)
rownames(type) = colnames(exprs0)
datosfenotipo = new("AnnotatedDataFrame", data = type)
sampleNames(datosfenotipo)
datosexperimento = new('MIAME',name='GSE64099',
    lab='Molecular Medicine Division, The Walter and Eliza Hall Institute
of Medical Research',
    contact ='mritchie@wehi.edu.au',
    title = '  Transcriptome profiling for genes transcriptionally
regulated by Smchd1 in lymphoma cell lines',
url = 'http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE64099')
gse64099  = new("ExpressionSet",exprs=exprs0,phenoData = datosfenotipo,
experimentData = datosexperimento,annotation = "MusMusculus")
```

Eliminamos los genes que no tienen ninguna lectura alineada. Todos estos genes no tienen interés cuando estudiemos la posible expresión diferencial.

```
nullsum = apply(exprs(gse64099),1,sum)==0
gse64099 = gse64099[!nullsum,]
```

6.13 GSE63776 a partir de conteos

En ocasiones tampoco es necesario realizar todo el procesado que hemos visto. Esto es, no necesitamos partir de las librerías de lecturas cortas y realizar todo el proceso de alineamiento. Los propios autores del estudio original han puesto a nuestra disposición las conteos. En este caso lo más cómodo es utilizar un `ExpressionSet` para almacenar la información. Como ejemplo los datos GSE63776.

1. Los bajamos de http://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE63776&format=file&file=GSE63776%5FPANC1%5Fcounts%2Etxt%2Egz.[130] Vamos a trabajar con los conteos.

2. Descomprimimos el fichero `GSE37211_count_table.txt.gz`.

3. Abrimos el fichero con un editor[131] y añadimos al principio de la primera línea la palabra `Gene`. Guardamos el fichero con la modificación.

4. Leemos los datos.

   ```
   x = read.table(file="GSE63776_PANC1_counts.txt",header=TRUE)
   ```

5. la primera columna es el nombre del gen por lo que la eliminamos para quedarnos con la matriz de expresión (conteos).

[130] Casi mejor poner el código de acceso GSE63776 en Google y tenemos la dirección.

[131] Bloc de nota, gedit, emacs

```
exprs0 = as.matrix(x[,-1])
rownames(exprs0) = x["Gene"]
```

6. Construimos los metadatos o datos fenotípicos.

```
type = factor(rep(1:2,each=3),levels=1:2,labels=c("siControl","siTCF7L2"))
pd0 = data.frame(type)
rownames(pd0) = colnames(exprs0)
datosfenotipo = new("AnnotatedDataFrame", data = pd0)
datosexperimento = new('MIAME',name='GSE63776',
    lab='Seth Frietze and Farnham P',
contact ='seth.frietze@unco.edu',
title = ' ',abstract = 'We have compared the genome-wide effects on the
transcriptome after treatment with ICG-001 (the specific CBP inhibitor)
versus C646, a compound that competes with acetyl-coA for the Lys-coA
binding pocket of both CBP and p300. We found that both drugs cause
large-scale changes in the transcriptome of HCT116 colon cancer cells and
PANC1 pancreatic cancer cells, and reverse some tumor-specific changes in
gene expression. Interestingly, although the epigenetic inhibitors affect
cell cycle pathways in both the colon and pancreatic cancer cell lines,
the WNT signaling pathway was affected only in the colon cancer cells.
Notably, WNT target genes were similarly down-regulated after treatment
of HCT116 with C646 as with ICG-001.',
url = 'http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE63776',
other = list(notes = 'Public on Jan 07, 2015'))
gse63776  = new("ExpressionSet",exprs=exprs0,phenoData = datosfenotipo,
experimentData = datosexperimento,annotation =
    "Illumina HiSeq 2000 (Homo sapiens)")
save(gse63776,file="gse63776.rda")
```

6.14 Normalización de RNA-seq

En esta sección se proponen distintos métodos de normalización propuestos en la literatura. Es cierto que los datos de RNA-seq tienen menos ruido que los datos de microarrays. En un principio se consideró que no se necesitaban procedimientos de normalización sofisticados [139]. No obstante, en el protocolo de preparación intervienen muchos procedimientos que introducen variabilidad: la extracción del RNA, la transcripción reversa, la amplificación y la fragmentacion.[132]

¿Cuáles pueden ser las covariables relativas a los genes y a las muestras que que pueden influir en los conteos y podamos considerar ruido técnico? [133]

Profundidad de secuenciación o tamaño de la librería [134] Estamos muestreando del total de moléculas disponibles. El tamaño de nuestra muestra es el total de lecturas. Este total de lecturas es distinto para las distintas muestras. En algunos procedimientos para análisis de la expresión diferencial este efecto no necesitará corrección previa pues el propio método incorpora en su modelo estocástico la distinta profundidad.

Composición del RNA Mediante la técnica RNA-seq tenemos una media de la abundancia relativa de cada gen en una muestra de

[132] En resumen, que no nos libramos de normalizar.

[133] El término ruido técnico es peligroso. ¿Cuándo es puramente técnico lo que eliminamos o cuando estamos eliminando señal biológica? Sinceramente el que escribe lo desconoce.

[134] Sequencing depth, size library.

RNA. Pero no tenemos una medida de la cantidad total de RNA por célula. Supongamos que tenemos una pequeña cantidad de genes que se expresan mucho en una muestra pero no en otra. En la muestra en donde se expresan mucho consumen una parte fundamental de la librería de lecturas de modo que otros genes que se expresen en esa muestra pero en mucho menor medida apenas aparecerán. De hecho, veremos que se expresan mucho menos de lo que realmente lo hacen porque los otros han consumido una parte importante del total de lecturas. En otras muestras donde no aparecen tan expresados esto no ocurrirá. Si no se ajusta la composición del RNA los otros genes aparecen falsamente infraregulados en la muestra donde unos pocos se expresan mucho. El método que vamos a utilizar para corregir este efecto es el la media ajustada de M-valores (TMM)[135] entre cada par de muestras.

[135] Trimmed mean of M-values (TMM).

Contenido GC El contenido GC de un gen no cambia entre muestras. Para un análisis de expresión diferencial no debiera influir. Algunos autores han indicado que puede que esto no sea tan cierto. La primera referencia en notar este efecto es [106].

Longitud del gen Genes más largos tenderán a tener más lecturas alineadas. Otra vez, entre muestras no cambia este factor y para un análisis de expresión diferencial no debiera tener influencia.

Sobre la longitud del gen. En [**OshlackWakefield09**] se comenta e ilustra el problema. ¿Qué efecto cabe experar cuando tenemos genes con distinta longitud. En RNA-seq lo que tenemos son fragmentos de transcritos. Si el transcrito es más largo el número de lecturas que producen es mayor. Es cierto que si queremos comparar entre muestras este efecto es el mismo. La longitud es la misma en todas las muestras. Sin embargo, al tener más lecturas la potencia del test es mayor para genes más largo y por ello encontraremos más genes con expresión diferencial entre los genes más largos. Un mayor tamaño muestral se asocia a una mayor probabilidad de rechazar la hipótesis nula de igualdad de las medias para una misma diferencia entre medias. De hecho, se propone en el trabajo indicado una simple formalización de este hecho. Denotamos por X la variable aleatoria que nos da el número de lecturas alineadas sobre un gen. Olvidande otros posibles sesgos podemos asumir que el valor medio de X depende del total de transcritos N y de la longitud del gen según la siguiente relación

$$\mu_X = EX = cNL,$$

siendo c un valor constante. En resumen asumimos que la media de lecturas es proporcional al tamaño de la librería y a la longitud del gen. Si se asume que la media y la varianza son iguales[136] entonces

$$var(X) = \sigma_X^2 = cNL,$$

[136] Lo cual es cierto bajo la hipótesis de que la variable X sigue una distribución de Poisson.

Si consideramos la variable X en dos muestras distintas tendremos X_i con $i = 1, 1$ y tamaños de librería N_i con $i = 1, 2$. Si utilizamos para contrastar la igualdad de medias el estadístico

$$\frac{X_1 - X_2}{\sqrt{cN_1L + cN_2L}}.$$

La potencia del test depende del cociente entre la media y la desviación estándar de $D = X_1 - X_2$[137] que viene dado por

[137] O error estándar.

$$\delta = \frac{cN_1L - cN_2L}{\sqrt{cN_1L + cN_2L}} \propto \sqrt{L}.$$

Es decir, δ es proporcional a la raiz cuadrada de la longitud del gen. Puede pensarse que un corrección simple como sería dividir el conteo por la longitud del gen corrige este efecto. Si aplicamos esta corrección no se resuelve el problema ya que entonces la diferencia sería $D' = \frac{X_1}{L} - \frac{X_2}{L}$. Como es bien conocido[138] tenemos que la media y varianza de X_i/N_i son

[138] § 3

$$E\frac{X_i}{L} = \frac{EX_i}{L} \quad \text{y} \quad var\left(\frac{X_i}{L}\right) = \frac{var(X_i)}{L^2}$$

Por tanto el cociente entre la media y la desviación estándar de D' no se modifica y vale

$$\delta = \frac{cN_1L - cN_2L}{\sqrt{cN_1L + cN_2L}} \propto \sqrt{L}.$$

6.14.1 RPKM

[139] El primer método de normalización se propuso en [98]. Supongamos que denotamos el conteo de interés por C. Es el número de lecturas alineadas sobre la característica de interés (exon, gen, etc.). El total de lecturas que podemos alinear es N (o tamaño de la librería). Y denotamos por L la longitud de la característica de interés en bp. Se define el RPKM como el siguiente cociente

[139] Reads per kilobase per million.

$$RPKM = \frac{10^9 C}{NL}. \tag{6.4}$$

Esta cuantificación de la expresión del gen nos permite comparar genes entre sí dentro de una misma muestra o librería.

En el paquete [154, geneLenDataBase] tenemos las longitudes de los distintos transcritos para distintos organismos. En el siguiente código leemos los datos correspondientes a humanos (versión 19).

```
pacman::p_load("geneLenDataBase")
data(hg19.ensGene.LENGTH)
head(hg19.ensGene.LENGTH)

##              Gene      Transcript Length
## 1 ENSG00000118473 ENST00000237247   4024
## 2 ENSG00000118473 ENST00000371039   4102
## 3 ENSG00000118473 ENST00000424320    964
## 4 ENSG00000118473 ENST00000320161   4693
## 5 ENSG00000118473 ENST00000371036   7294
## 6 ENSG00000118473 ENST00000407289   7901
```

6.14.2 Método TMM: Media ajustada de M-valores

Es, quizás, el método más popular de normalización. Fue propuesto en [115]. Comentan los autores en [115, pág. 2] el siguiente ejemplo

hipotético. Tenemos dos experimentos de secuenciación de RNA o muestras. Llamamos a estas dos muestras A y B. Supongamos dos conjuntos de genes disjuntos y con el mismo número de elementos: S_1 y S_2. El primer conjunto se expresa igualmente en A y B, es decir, tenemos el mismo número de transcritos en ambas muestras. Sin embargo, S_2 solamente se expresa en la muestra A y no en la muestra B. Además asumimos que no se expresa ningún otro gen. Si la profundidad de secuenciación es la misma en las dos muestras entonces el número de transcritos que observaremos de un gen de S_1 será la mitad en A que en B aunque sabemos que se expresa exactamente igual. En resumen, en la población (total de transcritos) el número total de transcritos de este gen es el mismo en ambas muestras. La probabilidad de observar uno de ellos depende de su frecuencia y del número de transcritos observados. Por ello, depende de que otros genes se expresen más o menos. En resumen, de la composición del RNA.

Sea X_{ij} (x_{ij}) el conteo aleatorio (observado) del gen i en la muestra j. Denotamos L_i la longitud del gen i y m_j es el total de lecturas de la librería j (o tamaño de la librería j). En [115] se asume que la media de X_{ij} verifica

$$E\Big[X_{ij}\Big] = \frac{\mu_{ij} L_i}{c_j} m_j,$$

siendo $c_j = \sum_{i=1}^{N} \mu_{ij} L_i$. Notemos que c_j nos está representando el total de RNA en la muestra. No se asume ninguna distribución de probabilidad específica para X_{ij}. La producción total de RNA, c_j, no es conocida. Sin embargo, sí es fácil estimar el cociente de estos valores para dos muestras, es decir, estimar el cociente $f_j = c_j/c_{j'}$.

Elegimos una muestra como muestra de referencia. Por ejemplo, la muestra r denota a partir de ahora la muestra tomada (arbitrariamente) como de referencia.

Nos fijamos en la muestra j y vamos a determinar la constante por la que multiplicaremos los conteos originales. En lo que sigue tanto j como r son fijos y las cantidades definidas dependen de i que denota el gen. Se define

$$M_{ij}^{(r)} = \log_2 \frac{x_{ij}/m_j}{x_{ir}/m_r} = \log_2(x_{ij}/m_j) - \log_2(x_{ir}/m_r),$$

y

$$A_{ij}^{(r)} = \frac{1}{2}\Big(\log_2(x_{ij}/m_j) + \log_2(x_{ir}/m_r) \Big)$$

Se eliminan los valores extremos tanto de los $M_{ij}^{(r)}$ como de los $A_{ij}^{(r)}$.[140] En concreto eliminamos un porcentaje de los M_i más pequeños y el mismo porcentaje de los más grandes. Lo mismo hacemos para los valores A_i. También eliminamos aquellos índices i tales que $x_{ij} = 0$ o bien $x_{ir} = 0$. El conjunto de índices i restante lo denotamos por G^*. Finalmente, el factor de normalización sería

[140] Recordemos que varía solamente i y mantenemos fijos j y r.

$$\log_2(TMM_j^{(r)}) = \frac{\sum_{i\in G^*} w_{ij}^{(r)} M_{ij}^{(r)}}{\sum_{i\in G^*} w_{ij}^{(r)}}$$

con

$$w_{ij}^{(r)} = \frac{m_j - x_{ij}}{m_j x_{ij}} + \frac{m_r - x_{ir}}{m_r x_{ir}}.$$

La muestra de referencia r es fija y lo que acabamos de calcular es el factor por el que multiplicamos los conteos originales de la muestra j. Este factor viene dado por $TMM_j^{(r)}$.[141]

Tanto este método como los que siguen están implementados en [33, edgeR].[142]

Utilizamos los datos tamidata::PRJNA297664 que es un

```
data(PRJNA297664,package="tamidata")
```

Accedemos a la matriz de expresión con los conteos.

```
pacman::p_load(SummarizedExperiment)
counts0 = assay(PRJNA297664)
```

```
pacman::p_load(edgeR)
deg.n =  DGEList(counts = counts0, group = rep(1,ncol(counts0)))
deg.n = calcNormFactors(deg.n, method = "TMM")
deg.n = estimateCommonDisp(deg.n)
counts1 =  deg.n$pseudo.counts
```

En la figura 6.2(a) y (b) tenemos la función de densidad y de distribución estimadas de los conteos antes de la normalización para las distintas muestras. Los dibujos se generan con el siguiente código.

```
pacman::p_load("ggplot2")
counts0 = data.frame(counts0)
df0 = reshape2::melt(counts0)
png(paste0(dirTamiFigures,"PRJNA297664_counts0_ecdf.png"))
ggplot(df0, aes(x=value, colour = variable)) + stat_ecdf()
dev.off()
png(paste0(dirTamiFigures,"PRJNA297664_counts0_ecdf.png"))
ggplot(df0, aes(x=value, colour = variable)) + geom_density()
dev.off()

counts1 = data.frame(counts1)
df0 = reshape2::melt(counts1)
png(paste0(dirTamiFigures,"PRJNA297664_counts0_ecdf.png"))
ggplot(df0, aes(x=value, colour = variable)) + stat_ecdf()
dev.off()
png(paste0(dirTamiFigures,"PRJNA297664_counts0_ecdf.png"))
ggplot(df0, aes(x=value, colour = variable)) + geom_density()
dev.off()
```

6.14.3 Mediana de los cocientes

Se propuso en [7]. Se consideran los factores de normalización

$$s_j = mediana_{\{i:\tilde{m}_i \neq 0\}} \frac{x_{ij}}{\tilde{m}_i} \tag{6.5}$$

siendo $\tilde{m}_i = \left(\prod_{j=1}^{n} x_{ij}\right)^{\frac{1}{n}}$. La idea es normalizar para cada gen diviendo el conteo por la correspondiente media geométrica de los

[141] Obviamente tenemos que $TMM_r^{(r)} = 1$ y esta muestra de referencia no se normaliza.

[142] El código relativo a métodos de normalización procede de [63, capítulo 4].

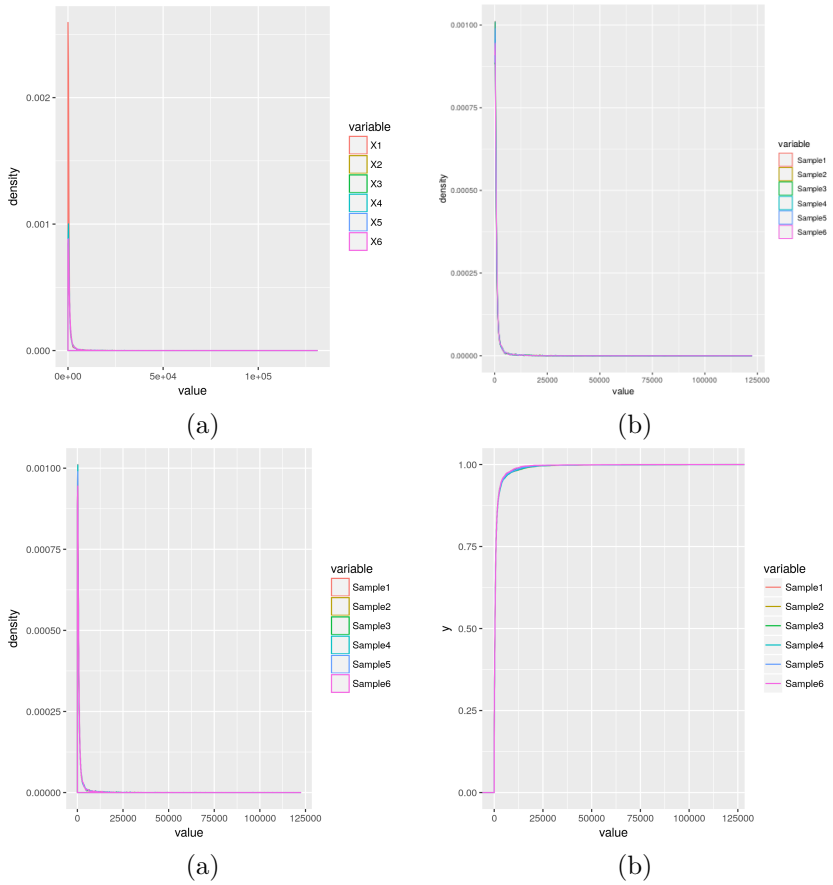


Figura 6.2: (a) Estimaciones de la función de densidad de los conteos para las distintas muestras de PRJNA297664 con los conteos originales. (b) Lo mismo que (a) con la función de distribución. En (c) y (d) tenemos los dibujos análogos utilizando los conteos después de aplicarles una normalización TMM.

distintos conteos para un mismo gen. Una vez que tenemos los datos normalizados intra gen buscamos una constante de normalización por muestra como la correspondiente mediana en la muestra. Estas constantes son utilizadas no para normalizar directamente el conteo sino para incorporarlo como factor que multiplica a la media del conteo en los métodos DESeq y DESeq2.

Utilizando los mismos datos tamidata::PRJNA297664 el método anterior lo podemos aplicar con el siguiente código.

```
deg.n =  DGEList(counts = counts0, group = rep(1,ncol(counts0)))
deg.n = calcNormFactors(deg.n, method = "RLE")
deg.n = estimateCommonDisp(deg.n)
counts2 =  deg.n$pseudo.counts
```

6.14.4 Homogeneizando los percentiles

Este método forma parte del método propuesto en [117] y comentado en §10.3. Indicamos aquí cómo aplicarlo a los datos tamidata::PRJNA297664. Como vemos en el código que sigue no se especifica método pues es el que aplica por defecto.

```
deg.n =  DGEList(counts = counts0, group = rep(1,ncol(counts0)))
deg.n = estimateCommonDisp(deg.n)
counts3 =  deg.n$pseudo.counts
```

6.15 Estudios de caso

6.15.1 SRP064411

[143] En esta sección vamos a obtener los datos, mapear las lecturas, contar y finalmente normalizar los conteos para los datos SRP064411. La descripción la podemos encontrar en http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE73681.

1. Los datos los podemos bajar de ftp://ftp-trace.ncbi.nlm.nih.gov/sra/sra-instant/reads/ByStudy/sra/SRP%2FSRP064%2FSRP064411/. También los podemos bajar de http://www.ebi.ac.uk/arrayexpress/experiments/E-GEOD-73681/?page=1&pagesize=500.

   Podemos bajar los datos accediendo a las direcciones indicadas y con el propio navegador. Suponemos que fijamos el directorio de trabajo donde tenemos los ficheros sra que acabamos de bajar.

2. Ejecutamos (y lo repetimos para cada fichero sra)

   ```
   fastq-dump -I --split-files SRR2549634.sra
   ```

3. Bajamos el fichero de índices para poder trabajar con Bowtie2. Nuestro datos corresponden a Sacaromices cerevisiae. En http://support.illumina.com/sequencing/sequencing_software/igenome.html los tenemos para distintas especies. En concreto, bajamos ftp://igenome:G3nom3s4u@ussd-ftp.illumina.com/Saccharomyces_cerevisiae/Ensembl/R64-1-1/Saccharomyces_cerevisiae_Ensembl_R64-1-1.tar.gz.

[143] Todo el análisis incluido en esta sección lo tenemos http://www.uv.es/ayala/docencia/tami/org/SRP064411.html y podemos reproducirlo ejecutando el código como se indica allí. No se puede reproducir exclusivamente con el material de esta sección.

4. Descomprimimos el fichero de índices y sacamos los ficheros.

```
gzip -d Saccharomyces_cerevisiae_Ensembl_R64-1-1.tar
    ↪ .gz
tar xvf Saccharomyces_cerevisiae_Ensembl_R64-1-1.tar
```

   Tenemos ahora un directorio `Saccharomyces\_cerevisiae_Ensembl_R64-1-1`.

5. Alienamos las lecturas. Para ello ejecutamos, para cada fichero fastq, lo siguiente sustituyendo los directorios por los correspondientes a nuestro caso. Supongamos que denotamos dirIndice donde tenemos los índices. En mi caso dirIndice es */DOCENCIA/tami-data/Saccharomyces_cerevisiae/Ensembl/R64-1-1/Sequence/Bowtie2Index/genome*. Ejecutamos lo siguiente sustituyendo dirIndice por el directorio correspondiente.

```
bowtie2 -x dirIndice -U SRR2549634_1.fastq -S
    ↪ SRR2549634_1.sam
```

6. Repetimos el punto anterior para cada uno de los ficheros fastq. Notemos que al tener lecturas simples (no apareadas) solamente tenemos un fichero fastq por muestra.

7. Se obtienen unos buenos porcentajes de lecturas alineadas superiores al 95% en todos los casos.

8. Transformamos de formato sam a formato bam ordenado utilizando samtools.

```
samtools view -bS SRR2549634_1.sam | samtools sort -
    ↪  SRR2549634_1
```

   Y como siempre repetimos para cada uno de los ficheros sam.

9. Ejecutamos el siguiente código que nos produce el SummarizedExperiment con el que podemos trabajar.

```
library(Rsamtools)
library(GenomicFeatures)
sampleTable = read.table("bamfiles.txt")
dirActualData =  "/home/gag/DOCENCIA/tami-data/SRP064411/sra/"
fls = paste(dirActualData,sampleTable[,1],sep="")
bamLst = BamFileList(fls, index=character(),yieldSize=100000,obeyQname=TRUE)
gtfFile = "~/DOCENCIA/tami-data/Saccharomyces_cerevisiae/Ensembl/R64-1-1/Annotatior
txdb = makeTxDbFromGFF(gtffile, format="gtf")
genes = exonsBy(txdb, by="gene")
library(GenomicAlignments)
SRP064411_SE = summarizeOverlaps(features = genes, read=bamLst,
    mode="Union",
    singleEnd=TRUE, ## No son lecturas apareadas
    ignore.strand=TRUE,
    fragments=FALSE)
```

10. Tenemos que añadir los metadatos o datos fenotípicos. Son los siguientes donde lo que aparece en la primera columna es el nombre que tiene la muestra en nuestro `SummarizedExperiment`, los `colnames`.

	SampleName	Run	Treatment	Rep
SRR2549634_1.bam	GSM1900735	SRR2549634	0	1
SRR2549636_1.bam	GSM1900737	SRR2549636	0	3
SRR2549638_1.bam	GSM1900739	SRR2549638	1	2
SRR2549635_1.bam	GSM1900736	SRR2549635	0	2
SRR2549637_1.bam	GSM1900738	SRR2549637	1	1
SRR2549639_1.bam	GSM1900740	SRR2549639	1	3

Creamos un `DataFrame`[144] con estas variables.

[144] Que no es un data.frame.

```
SampleName = c("GSM1900735","GSM1900737","GSM1900739","GSM1900736",
    "GSM1900738","GSM1900740")
Run = c("SRR2549634","SRR2549636","SRR2549638","SRR2549635",
    "SRR2549637","SRR2549639")
Treatment = factor(c(0,0,1,0,1,1),levels=0:1,labels=c("wild-type","sec66del"))
Rep = c(1,3,2,2,1,3)
colData(SRP064411_SE) = DataFrame(SampleName,Run,Treatment,Rep)
save(SRP064411_SE,file="SRP064411_SE.rda")
```

Los datos `SRP064411_SE` los tenemos en [10, tamidata].

6.16 Datos simulados

En esta sección enumeramos algunos simuladores de datos RNA-Seq. Una dirección de interés es https://omictools.com/data-simulation3-category.

Parte III

Expresión diferencial

Capítulo 7

Expresión diferencial marginal

7.1 Introducción

Para cada uno de los genes (exones, o en general características genómicas) tenemos un valor numérico (expresión) que nos indica abundancia de copias de esta característica. En resumen abundancia de la característica en la que estamos interesados. Tenemos este valor observado para distintas muestras. De cada una de las muestras tenemos a su vez variables que las describen: tiempos, tamaños celulares, temperatura, etc. A estas variables las llamaremos en lo que sigue covariables. Estas covariables pueden ser categóricas, numéricas o tiempos de supervivencia censurados. Si la covariable tiene dos categorías entonces nos define dos grupos (por ejemplo, control y tratamiento). El caso en que queremos comparar dos grupos es el más frecuente y por ello siempre le daremos una mayor atención.

El problema que se conoce como **expresión diferencial** se puede traducir a saber si hay algún tipo de asociación entre las expresiones observadas y los valores de la covariable. Si la covariable define dos categorías entonces la pregunta de la asociación covariable-expresión se puede formular como: ¿Hay diferencias entre la expresión de genes entre dos grupos considerados? Dicho de otro modo, la expresión es distinta bajo los tratamientos que estamos considerando.

En un primer momento vamos a adoptar la aproximación gen-a-gen. Buscamos genes que se expresan diferencialmente sin atender a las interacciones que puedan existir entre los distintos genes, que las hay y las consideraremos más adelante.

Intentaremos ir planteando las cuestiones (muy) lentamente ilustrando continuamente lo que hacemos con R/Bioconductor.

7.2 Algo de notación

En lo que sigue denotaremos la matriz con los datos de expresión de los distintos genes como $\boldsymbol{x} = [x_{ij}]_{i=1,\ldots,N;j=1,\ldots,n}$ donde el valor x_{ij} nos da el nivel de expresión **observado** del gen i-ésimo en la muestra j-ésima. Asociada a la muestra j tenemos una covariable (o variable fenotípica) y. El valor de la covariable y en la muestra j la vamos

adenotar por y_j.[1]

Como es de uso habitual en Estadística denotaremos

$$x_{\cdot j} = \sum_{i=1}^{N} x_{ij}; \quad x_{i\cdot} = \sum_{j=1}^{n} x_{ij},$$

$$\bar{x}_{\cdot j} = \sum_{i=1}^{N} \frac{x_{ij}}{N}; \quad \bar{x}_{i\cdot} = \sum_{j=1}^{n} \frac{x_{ij}}{n}.$$

En la notación previa hemos indica los valores de expresión o la covariable utilizando letras en minúsculas. Esto es habitual en Probabilidad. Sin embargo cuando consideremos los valores que observaremos **antes** de realizar la experimentación tendremos los valores aleatorios. También como es usual utilizaremos las letras en mayúsculas. De modo que $\boldsymbol{X} = [X_{ij}]_{i=1,\ldots,N;j=1,\ldots,n}$ denota la matriz de expresión aleatoria cuando todavía no se ha realizado la experimentación.

7.3 Selección no específica o filtrado independiente

El material de esta sección se ha obtenido de [65, capítulo 5] y [54]. Como material complementario es muy interesante [20].

La mayor parte de los genes no se expresan de un modo diferenciado entre condiciones. Y hay muchos.[2] Más genes evaluados *simultáneamente* suponen una dificultad mayor en el tratamiento estadístico posterior. Por ello si podemos de un modo simple *quitar* genes que no parecen tener actividad pues mucho mejor.[3]

Es costumbre realizar una preselección, un filtrado que no utilice información de la covariable y. Utilizamos el perfil de expresión del gen pero no información sobre las muestras, la posible clasificación de muestras en distintos grupos. A este filtrado se le da en la literatura la denominación de *filtrado independiente* o bien **selección no específica**.

Veamos un posible método. Consideramos el gen i-ésimo y tomamos una medida de localizacion como la mediana. Denotamos este valor por u_i. Determinamos un cierto percentil de estos valores u_i. Por ejemplo, si denotamos por p_1 el orden del percentil entonces $q_{p_1}(u)$ sería el correspondiente percentil de orden p_1 de los valores u_i. Mantendremos los genes tales que su valor u_i sea mayor que $q_p(u)$. Ahora consideramos una medida de dispersión como el coeficiente intercuartílico. Del mismo modo, v_i sería el coeficiente intercuartílico del i-ésimo gen y el correspondiente percentil sería $q_{p_2}(v)$ donde el orden p_2 no tiene porqué coincidir con p_1. Nos quedamos con los genes en el siguiente conjunto

$$\{i : i = 1, \ldots, N; u_i \geq q_{p_1}(u); v_i \geq q_{p_2}(v)\},$$

[1] Por ejemplo, si estamos comparando los niveles de expresión en dos grupos (un grupo control y un grupo de enfermos por ejemplo) entonces y_j tomará el valor 1 si está en el primer grupo (control) y valor 2 si está en el segundo grupo (valor 2).

[2] A veces uno piensa que demasiados.

[3] Aunque esto puede tener muchos riesgos como descartar genes que actúan conjuntamente con otros y marginalmente no tengan una actividad apreciable. Es pues algo peligroso eliminar genes sin tener muy claro como lo hacemos.

es decir, aquellos que tienen un nivel de expresión alto y una variabilidad alta. En otras palabras, genes que se expresan y que lo hacen de un modo variable en las distintas muestras.

En el método que acabamos de explicar podemos sustituir la mediana como localización y el rango intercuartílico como dispersión por la media y la desviación estándar.

Otro procedimiento que se ha propuesto y utilizado con frecuencia es que un gen se exprese con claridad en algunas muestras, lo que podemos llamar el método k-sobre-A. La idea es simple, un gen lo conservamos si en al menos k muestras tiene un nivel de expresión por encima de un valor mínimo A.

En lo que sigue vemos cómo utilizar estas ideas con R/Bioconductor. En § 7.3.1 se propone una opción muy sencilla para preseleccionar genes utilizando simplemente la función *apply*. En § 7.3.2 utilizamos el paquete [54, genefilter]. Finalmente en § 7.3.3 utilizamos la función genefilter::nsFilter.

Vamos a abordar el problema con los datos ALL::ALL. En primer lugar aplicamos la selección de muestras de § 5.4.

```
pacman::p_load("Biobase","ALL")
data(ALL)
bcell = grep("^B",as.character(ALL$BT))
types = c("NEG","BCR/ABL")
moltyp = which(as.character(ALL$mol.biol) %in% types)
bcrneg = ALL[,intersect(bcell,moltyp)]
```

7.3.1 Utilizando apply

Vamos a implementar un par de procedimientos de selección independiente con la función `apply`.

Un procedimiento muy básico puede ser calcular para el rango intercuartílico del perfil de expresión de cada gen.

```
bcrneg.iqr = apply(exprs(bcrneg),1,IQR)
```

Y luego podemos determinar el cuantil (por ejemplo, el percentil 0.5 o mediana) de los rangos intercuartílicos. Si estás por encima de este percentil conservamos el gen. De lo contrario, no lo consideramos en lo que sigue.

```
sel.iqr = (bcrneg.iqr > quantile(bcrneg.iqr,0.5))
```

Una idea como la que acabamos de aplicar para seleccionar genes tiene el inconveniente de que si un grupo tiene pocos datos entonces aunque hubiera una expresión diferencial la variabilidad total no tiene porqué ser muy grande.

Ahora filtramos atendiendo a la mediana del perfil de expresión.

```
bcrneg.median = apply(exprs(bcrneg),1,median)
sel.median = (bcrneg.median > quantile(bcrneg.median,0.5))
```

Y nos quedamos con los genes que verifican ambas condiciones. Nuestro ExpressionSet original es **bcrneg**, ¿cómo nos quedamos con estos genes y todas las muestras? Con el siguiente código.

```
bcrneg1 = bcrneg[sel.iqr & sel.median,]
```

Media y desviación estándar

Sustituimos mediana y rango intercuartílico por media y desviación estándar respectivamente.

Es claro que podríamos sustituir la desviación estándar por alguna otra medida de variabilidad, por ejemplo, el rango intercuartílico. El método anterior se podría implementar con el siguiente código.

```
bcrneg.sd = apply(exprs(bcrneg),1,sd)
sel.sd = (bcrneg.sd > quantile(bcrneg.sd,0.5))
bcrneg.mean = apply(exprs(bcrneg),1,mean)
sel.mean = (bcrneg.mean > quantile(bcrneg.mean,0.5))
bcrneg2 = bcrneg[sel.sd & sel.mean,]
```

k sobre A

Otra opción natural sería fijar un nivel de actividad mínima para un gen y quedarnos con aquellos que superen este nivel mínimo de actividad. Consideremos el siguiente criterio: si el nivel de expresión mínimo es c y tenemos n muestras podemos pedir que un gen determinado se considere activo si en al menos k muestras del total de n su nivel de expresión supere este nivel mínimo de actividad.

¿Cómo hacerlo con R? Empezamos fijando el nivel mínimo c. ¿Y cómo? Podemos ver los percentiles de todas las expresiones (todos los genes y todas las muestras).

```
quantile(exprs(bcrneg))
```

```
##        0%       25%       50%       75%      100%
##  1.984919  4.117796  5.468801  6.832439 14.044896
```

Nos quedamos con la mediana para c.

```
c =  5.468801
```

Determinamos qué niveles de expresión lo superan.

```
overc  = exprs(bcrneg) > c
```

Podemos ver los resultados para la fila 433.

```
overc[433,]
```

Determinamos el número de muestras por fila que superan el valor c. Notemos que aplicamos una suma. Cuando sumamos un vector lógico (con valores TRUE y FALSE) entonces el valor TRUE se interpreta como el número 1 y el valor FALSE se interpreta como 0.

```
count.c = apply(overc,1,sum)
```

Por ejemplo, las primeras filas son

```
head(count.c)

##   1000_at   1001_at 1002_f_at 1003_s_at   1004_at
##        79         3         0        77        76
##   1005_at
##        79
```

Y finalmente determinamos si estamos por encima de 5 muestras.

```
sel.c = count.c >= 5
```

Y veamos cuántos genes conservamos.

```
table(sel.c)

## sel.c
## FALSE  TRUE
##  4736  7889
```

Una vez elegido un procedimiento de los comentados eliminamos aquellos genes que no pasan el criterio de selección. Y con los genes que nos quedan seguiríamos estudiando si hay o no expresión diferencial. Observemos que no hemos utilizado para nada los grupos que pretendemos comparar.

7.3.2 Utilizando genefilter

En la anterior sección utilizamos funciones básicas de R para realizar la selección no específica. Podemos utilizar un paquete diseñado para esto. El paquete genefilter [54] nos permite implementar distintos procedimientos de filtrado independiente de un modo sencillo.

Seguimos utilizando los datos del ExpressionSet **bcrneg**. Supongamos un procedimiento de filtrado no específico tan simple como el siguiente: mantendremos aquellos genes tal que su nivel de expresión es al menos de 200 en al menos 5 de las muestras. Para hacer esto con el paquete [54, genefilter] hemos de realizar tres pasos:

1. Crear la función que implementa cada criterio de filtrado.
2. Combinar todos los criterios de filtrado en una única función.
3. Aplicar la función de filtrado final a la matriz de expresión.

El siguiente código implementa todo el procedimiento.

```
library(Biobase)
library(genefilter)
f1 = kOverA(5, 5.468801)
ffun = filterfun(f1)
wh1 =  genefilter(exprs(bcrneg), ffun)
```

Finalmente podemos ver cuántos genes permanecen en nuestro estudio. Son aquellos que tiene el valor de wh1 igual a TRUE.

```
table(wh1)

## wh1
## FALSE  TRUE
##  4736  7889
```

Vamos a incorporar los otros dos criterios comentados previamente. Primero necesitamos una función que nos diga si la desviación estándar supera el valor c.

Primero vemos el modelo que nos ofrece la función `genefilter::kOverA()`.

```
kOverA = function (k, A = 100, na.rm = TRUE)
{
    function(x) {
        if (na.rm)
            x = x[!is.na(x)]
        sum(x > A) >= k
    }
}
```

Y nos definimos algo similar utilizando la desviación estándar.

```
sdOverc = function (c, na.rm = TRUE)
{
    function(x) {
        if (na.rm)
            x = x[!is.na(x)]
        sd(x) >= c
    }
  }
```

Y ahora otra función que evalúe si el rango intercuartílico supera un cierto valor.

```
iqrOverc = function (c, na.rm = TRUE)
{
    function(x) {
        if (na.rm)
            x = x[!is.na(x)]
        IQR(x) >= c
    }
  }
```

```
f1 = kOverA(5, 200)
f2 = sdOverc(150)
f3 = iqrOverc(72)
ffun = filterfun(f1,f2,f3)
wh123 =  genefilter(exprs(bcrneg), ffun)
```

¿Qué genes verifican este criterio de preselección?

```
table(wh123)

## wh123
## FALSE
## 12625
```

7.3.3 Utilizando nsFilter

Supongamos que queremos aplicar la siguiente selección nos vamos a quedar con aquellas sondas tales que el rango intercuartílico (para todas las muestras) es mayor que la mediana de los rangos intercuartílicos. Luego vamos a necesitar conocer la anotación de los genes. Si esto es así podemos filtrar aquellos genes de los cuales desconocemos su anotación. Para ello necesitamos conocer primero el paquete de anotación que utilizan nuestros datos.[4]

```
annotation(ALL)

## [1] "hgu95av2"
```

Por tanto hemos de cargar este paquete.

```
library(hgu95av2.db)
```

Realizamos el filtrado correspondiente.[5]

```
bcrneg.filt1 = nsFilter(bcrneg,var.func=IQR,var.cutoff=0.5,
    require.GOBP=TRUE)
```

La función que utilizamos para medir variabilidad podemos ir modificándola. Por ejemplo, podemos sustituir `var.func` por la varianza o por la desviación estándar. Así podríamos realizar

```
bcrneg.filt2 = nsFilter(bcrneg,var.func=sd,var.cutoff=0.5,
    require.GOBP=TRUE)
```

¿Cómo podemos combinar ambas selección? Utilizando la función `genefilter::nsFilter()` tenemos un `Biobase::ExpressionSet`. Una manera simple puede ser la siguiente.

```
sel = intersect(featureNames(bcrneg.filt1),
    featureNames(bcrneg.filt2))
```

Y nuestro `ExpressionSet` filtrado sería

```
bcrneg1 = bcrneg[sel,]
```

Y ya podemos seguir trabajando con los genes que hemos filtrado.

7.4 Fold-change

[145] Queremos comparar dos grupos. Denotamos los valores de expresión originales con x_{ij} (respectivamente y_{ij}) para la i-ésima característica en la j-ésima muestra del primer grupo (respectivamente del segundo grupo). A los logaritmos (en base 2 o log2) de los valores originales los denotamos por $u_{ij} = \log_2(x_{ij})$ y $v_{ij} = \log_2(y_{ij})$. Para cada

[145] Lo que comentamos en esta sección me ha costado de entender una enfermedad. Lo primero el porqué de utilizarlo y luego que hay una indefinición en el término en la literatura. Vamos a intentar aclarar qué significa antes de que se me olvide. Porque mucho interés en usarlo no tengo.

[4] En http://www.bioconductor.org/packages/release/data/annotation/ tenemos un listado de paquetes de anotación.

[5] Observemos que ahora no pasamos la matriz de expresión sino el ExpressionSet mismo.

gen, tendremos una expresión media para la i-ésima característica en cada uno de los dos grupos $\bar{x}_{i\cdot} = \sum_{j=1}^{n_1} \frac{x_{ij}}{n_1}$ para la media en el primer grupo. Similarmente definimos $\bar{y}_{i\cdot}$, $\bar{u}_{i\cdot}$, $\bar{v}_{i\cdot}$. ¿Qué se entiende por **fold-change**? Dos son las interpretaciones de este valor. La primera ([136]) lo define como

$$FC_i^{(1)} = \frac{\bar{x}_{i\cdot}}{\bar{y}_{i\cdot}}. \tag{7.1}$$

Lo definimos como el cociente de las medias de las expresiones en la escala original. La segunda definición (que no es equivalente a la primera) es

$$FC_i^{(2)} = \bar{u}_{i\cdot} - \bar{v}_{i\cdot}, \tag{7.2}$$

tenemos la diferencia de las medias de las log2 expresiones.

[146] Que no es estadístico y que por tanto no vamos a considerar en este manual.

Un procedimiento[146] que se ha utilizado frecuentemente en la literatura de microarrays consiste en tomar el log2 del fold-change en la definición 7.2, el valor

$$\log_2 FC_i^{(1)} = \log_2 \left(\frac{\bar{x}_{i\cdot}}{\bar{y}_{i\cdot}} \right).$$

Si el módulo del cociente anterior es mayor que una constante positiva c ($c > 0$) entonces diríamos que el gen i se sobre expresa en el grupo 1 en relación con el grupo 2 ya que

$$\log_2 \left(\frac{\bar{x}_{i\cdot}}{\bar{y}_{i\cdot}} \right) \geq c.$$

y se sigue que

$$\frac{\bar{x}_{i\cdot}}{\bar{y}_{i\cdot}} \geq 2^c.$$

Si $\log_2 \left(\frac{\bar{x}_1}{\bar{x}_2} \right) < -c$ entonces tendremos que

$$\frac{\bar{x}_{i\cdot}}{\bar{y}_{i\cdot}} \leq -2^c,$$

o bien que

$$\frac{\bar{y}_{i\cdot}}{\bar{x}_{i\cdot}} \geq 2^c.$$

En este segundo caso el gen i se sobre expresa en el grupo 2 en relación con el primer grupo. El término que se utiliza es sobre regulación en el primer caso y de infra regulación en el segundo.[147] Se utiliza con mucha frecuencia. Es sencillo, entendible y fácil de usar. Pero **no** es una buena opción. Lo fundamental, *no se tiene en cuenta la variabilidad de las medias* que estamos comparando. Una opción más correcta la vemos posteriormente en este tema y es la utilización de un test de la t para comparar las medias de las dos poblaciones que estamos comparando. O versiones modificadas del t-test clásico como la propuestas en [136] o en [125] en donde esencialmente se modifica la estimación del error estándar.

[147] Up (down) regulation.

7.5 Los peligros de la selección no específica

En este tema hemos presentado como habitual la combinación de una selección no específica o independiente de genes con un análisis

posterior marginal de los genes que no han sido seleccionados utilizando procedimientos de corrección por comparaciones múltiples. Este procedimiento, aún habitual en la literatura, tiene sus riesgos. Un trabajo de gran interés a consultar sobre este problema es [20].

7.6 Expresión diferencial de un solo gen

Seguimos trabajando con los datos golub. Cargamos los datos.

```
library(Biobase)
data(gse21942,package="tamidata")
```

Vamos a comparar la expresión observada de un gen para enfermos (esclerosis múltiple) y para sanos. La variable fenotípica que nos indica si la muestra corresponde a enfermo o sano es

```
y0 = pData(gse21942)[,"FactorValue..DISEASE.STATE."]
```

Pretendemos fijarnos en un gen y ver cómo comparamos los niveles de expresión. En concreto nos fijamos en el gen que ocupa la primera fila de la matriz de expresión. Esta es la información que tenemos del mismo.

```
fData(gse21942)[1,]

##     PROBEID ENTREZID         ENSEMBL SYMBOL
## 1 1007_s_at      780 ENSG00000204580   DDR1
##           GO EVIDENCE ONTOLOGY
## 1 GO:0001558      IEA       BP
```

Guardamos los niveles de expresión en *DDR1*.

```
DDR1 = exprs(gse21942)[1,]
```

En la figura 7.1(a) mostramos los datos para las distintas muestras (con distintos colores y caracteres distintos).

```
pacman::p_load("ggplot2")
df=data.frame(id = 1:ncol(gse21942),expression = DDR1,type = golub.cl)
ggplot(df,aes(x= id,y = expression,color = type)) + geom_point()
```

Sabemos que hay dos grupos de pacientes. Podemos representar dos diagramas de cajas que nos den una comparación sencilla de los niveles de expresión para los dos grupos. Lo tenemos en figura 7.1(b).

```
df=data.frame(id = 1:ncol(gse21942),expression = DDR1,type = y0)
ggplot(df,aes(x= id,y = expression,color = type)) + geom_boxplot()
```

También trabajaremos con las expresiones de THRA que aparece en la fila 7 de la matriz de expresión de tamidata::gse21942.

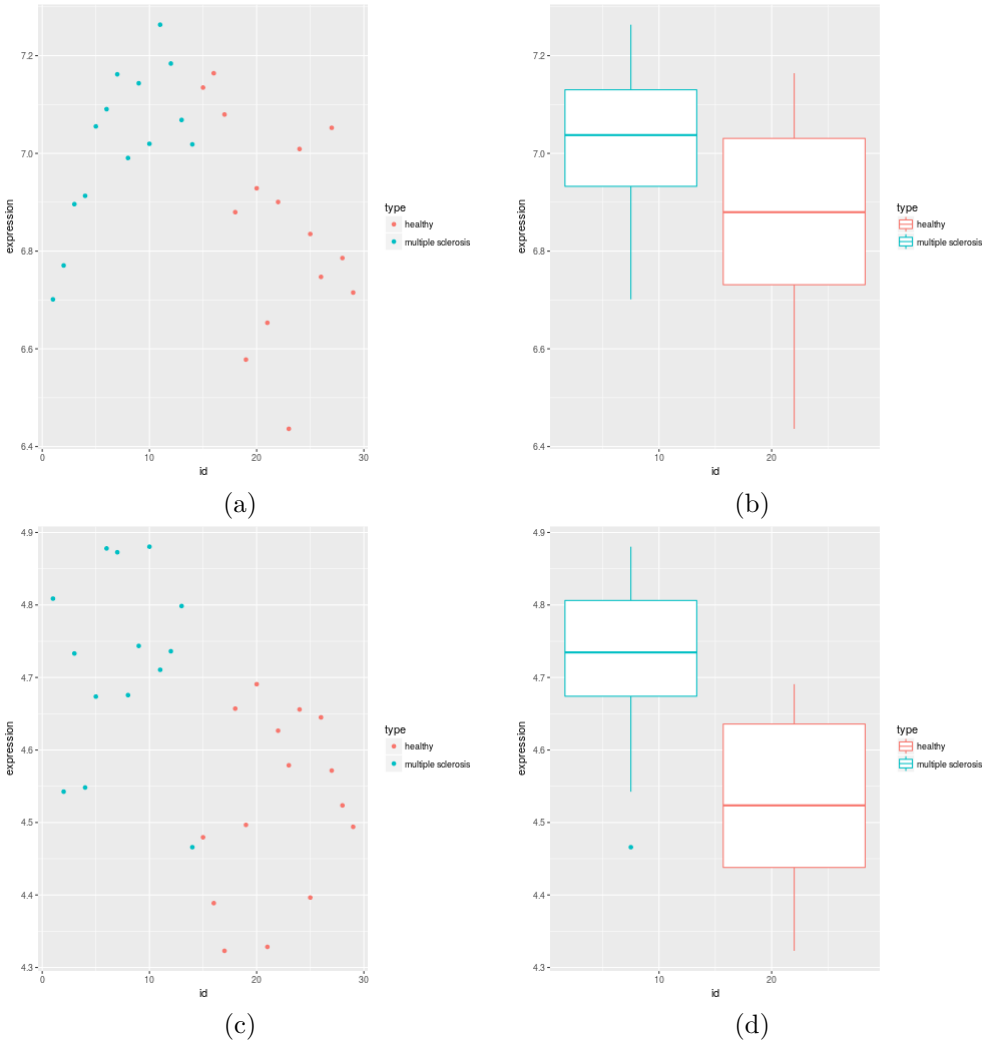


Figura 7.1: a) Perfil de expresión para el gen DDR1 utilizando casos y controles. b) Diagrama de cajas para gen DDR1 en los dos grupos. c) Perfil de expresión para el gen THRA en los dos grupos. d) Diagrama de cajas para gen DDR1 en los dos grupos.

```
fData(gse21942)[7,]

##     PROBEID ENTREZID         ENSEMBL SYMBOL
## 332 1316_at     7067 ENSG00000126351   THRA
##             GO EVIDENCE ONTOLOGY
## 332 GO:0000976      IEA       MF
```

Guardamos los datos en `THRA`.

```
THRA = exprs(gse21942)[7,]
```

En las figuras 7.1(c) y 7.1(d) mostramos el perfil de expresión de este gen diferenciando las expresiones según el tipo de leucemia y el correspondiente diagrama de cajas.

```
df=data.frame(id = 1:ncol(gse21942),expression = THRA,type = y0)
ggplot(df,aes(x= id,y = expression,color = type)) + geom_point()
ggplot(df,aes(x= id,y = expression,color = type)) + geom_boxplot()
```

Este es el problema planteado: dos grupos de valores indicando los niveles de expresión bajo dos condiciones y la pregunta fundamental a responder es: ¿son ambos grupos de valores, ambos grupos de niveles de expresión, similares entre sí o difieren claramente uno de otro? Estamos con lo que en Estadística se conoce como la comparación de dos poblaciones: cada población corresponde con una condición.

7.7 Comparamos dos condiciones

Tenemos datos relativos a niveles de expresión bajo dos condiciones experimentales. En nuestro caso, la condición experimental me indica el tipo de leucemia. Denotamos por X el nivel de expresión aleatorio que observamos bajo la primera condición y por Y lo mismo pero con la segunda condición.

Supongamos que asumimos que ambos tienen una distribución normal con medias y varianzas posiblemente distintas. Es decir, asumimos que $X \sim N(\mu_X, \sigma_X^2)$ y que $Y \sim N(\mu_Y, \sigma_Y^2)$. También asumimos que tenemos una muestra aleatoria de X: $X_1 \ldots, X_n$ variables aleatorias independientes y con una misma distribución. Y otra muestra de Y: $Y_1 \ldots, Y_m$ variables aleatorias independientes y con la distribución. En nmr13-§ 7 se estudia con detalle los contrastes de hipótesis de comparación de medias de dos poblaciones normales. Utilizamos los datos `tamidata::gse21942`.

```
data(gse21942,package="tamidata")
```

Utilizamos `genefilter::rowttests()`.

```
tt = genefilter::rowttests(gse21942,pData(gse21942)[,"FactorValue..DISEASE.STATE."])
```

Veamos las primeras filas de la salida.

```
head(tt)
```

```
##             statistic          dm     p.value
## 1007_s_at -2.29869931 -0.15981906 0.029492008
## 1053_at    3.44084102  0.20940361 0.001901206
## 117_at    -0.08505071 -0.01110444 0.932848609
## 121_at    -0.53362792 -0.02274832 0.597965094
## 1255_g_at -1.01536731 -0.04339509 0.318943716
## 1294_at   -1.05030996 -0.07873761 0.302886126
```

Cada fila corresponde a un gen. La primera columna (`statistic`) nos muestra el estadístico t del contraste de hipótesis para la igualdad de las medias (por defecto asume varianzas iguales). La segunda columna (`dm`) nos muestra la diferencia de medias y la última columna (`p.value`) nos da el p-valor del contraste.

Para cada gen tenemos pues un p-valor. ¿Podemos seguir aplicando el criterio de rechazar la hipótesis nula cuando el p-valor es inferior al nivel de significación α previamente elegido?

Si lo hacemos en este ejemplo, ¿cuándo tests mostrarían una expresión diferencial entre ambos grupos?

```
table(tt[,"p.value"] < 0.05)

##
## FALSE  TRUE
## 37561 17114
```

Como vemos tenemos demasiados genes que muestran expresión diferencial. Y no parece muy razonable esto. ¿Con qué problema nos estamos encontrando? El nivel de significación α es una cota superior del error tipo I cuando realizamos **un solo test**. Pero no estamos aplicando un solo test. En concreto con estos datos acabamos de realizar 3051 tests que además son dependientes entre si. Estamos rechazando muchas hipótesis nulas (no diferencia entre grupos para cada gen) cuando no deberíamos de hacerlo.

Sin embargo, cuando realizamos muchos test: ¿qué es el error tipo I? Ya no es claro cómo cuantificamos los errores que cometemos y, de hecho, tenemos que definir nuevas tasas de error para cuantificar y, sobre todo, controlar los errores que cometemos cuando realizamos *simultáneamente* muchos tests.

Ej. 17 — Utilizando los datos tamidata::gse1397 y el factor `pData(gse1397[,"type"]` realizar un análisis de expresión diferencial marginal

7.8 Comparando más de dos condiciones

En todo lo anterior hemos considerado la situación más habitual. Tenemos las muestras clasificadas en dos grupos o condiciones y pretendemos determinar aquellos genes que se expresan de un modo distinto entre ambas condiciones, que se expresan diferencialmente. Sin embargo, podemos tener más de dos grupos y pretender realizar un análisis similar. Esto nos conduce al análisis de la varianza (o abreviadamente anova). Vamos a recordar brevemente qué es y cómo se aplica un análisis anova.

Supongamos que tenemos I condiciones distintas y en cada una de ellas n_i muestras de modo que $\sum_{i=1}^{I} n_i = n$, el total de muestras de

las que disponemos. Supongamos que Y_i denota la expresión aleatoria de un gen bajo la condición i. Se asume que

$$Y_i \sim N(\mu_i, \sigma^2),$$

es decir, que dicha expresión aleatoria se distribuye como una normal con media dependiente (posiblemente) de la condición pero con una varianza común σ^2 .[148] La no existencia de expresión diferencial entre las condiciones se traduce, asumiendo este modelo probabilístico, en que las expresiones medias son la misma. En definitiva, *no existe expresión diferencial* es equivalente (bajo el modelo) a

[148] Recordemos que en el caso de dos grupos podemos asumir o no una misma varianza.

$$H_0 : \mu_1 = \ldots = \mu_I, \tag{7.3}$$
$$H_1 : Existe\ i \neq j\ \mu_i \neq \mu_j. \tag{7.4}$$

Supongamos que, para un gen dado, denotamos por y_{ij} la j-ésima muestra observada bajo la condición i ($i = 1, \ldots, I$ y $j = 1, \ldots, n_i$). Se consideran las *sumas de cuadrado intra*

$$SS(Intra) = \sum_{i=1}^{I} \sum_{j=1}^{n_i} (y_{ij} - \bar{y}_{i\cdot})^2$$

y la *suma de cuadrados entre* como

$$SS(Entre) = \sum_{i=1}^{I} n_i (\bar{y}_{i\cdot} - \bar{y}_{\cdot\cdot})^2$$

El estadístico para contrastar esta hipótesis nula es

$$F = \frac{SS(Entre)/(I-1)}{SS(Intra)/(n-I)}$$

Bajo la hipótesis nula de que todas las medias son la misma (y puesto que asumimos una misma varianza) tendríamos una distribución común bajo todas las condiciones. Asumiendo la hipótesis nula el estadístico F se distribuye como un F con I-1 y n-I grados de libertad,

$$F \sim F_{I-1, n-I}.$$

Ejemplo 7.1 (Anova con R) *Lo ilustramos la expresión bajo diferentes condiciones de un gen de los datos de expresión gse20986 (§ 5.11).*

```
data(gse20986,package="tamidata")
```

Seleccionamos un gen cualquiera.

```
y = exprs(gse20986)[678,]
```

y realizamos un análisis de la varianza.

```
summary(aov(y ~ pData(gse20986)[,"tissue"]))
```

```
##                             Df   Sum Sq  Mean Sq
## pData(gse20986)[, "tissue"]  3 0.005309 0.001770
## Residuals                    8 0.020212 0.002527
##                             F value Pr(>F)
## pData(gse20986)[, "tissue"]     0.7  0.578
## Residuals
```

De acuerdo con el p-valor observado no rechazaríamos la hipótesis nula a ninguno de los niveles de significación habituales (0.05 o 0.01).

En el siguiente ejemplo analizamos, buscando diferencias entre cualquier par de grupos (tejidos en el ejemplo).

Ejemplo 7.2 (GSE20986) *Vamos a realizar un análisis de expresión diferencial de los datos gse20986. Cargamos paquetes necesarios*

```
library(multtest)
library(genefilter)
```

Tenemos cuatro grupos. Tenemos interés en la comparación entre todos los grupos (tejidos) al mismo tiempo. En definitiva buscamos aquellos genes que muestran una expresión diferencial entre al menos dos de los grupos. Un posible planteamiento es el modelo de análisis de la varianza y la comparación entre los grupos se reformula como una comparación entre valores medios.[6] *Esto supone una comparación simultánea de las medias de los cuatro grupos y esto se realiza con un análisis de la varianza.*

Calculamos los p-valores con la función genefilter::rowFtests.

```
gse20986.aov =
    rowFtests(gse20986, pData(gse20986)[,"tissue"])
```

Por ejemplo, vemos los dos primeros valores.

```
head(gse20986.aov,n=2)

##           statistic     p.value
## 1007_s_at  9.782456 0.004715281
## 1053_at   14.144595 0.001455658
```

7.9 Ejercicios

Ej. 18 — Con los datos multtest::golub y utilizando las funciones base::apply, stats::sd, stats::IQR se pide:

1. Determinar para cada gen (esto es, las expresiones de una fila dada) el rango intercuartílico.
2. Una vez hemos calculado el rango intercuartílico para gen calcular el percentil de orden 0.50 (o mediana).
3. Determinar las filas correspondientes a los genes cuyo rango intercuartílico supera el percentil que hemos calculado en el apartado anterior.
4. Determinar aquellas filas donde al menos 10 muestras de las 38 superan una expresión de 0.
5. Seleccionar en la matriz de expresión golub las filas que verifican los criterios dados en los puntos 3 y 4.

[6] No es la única opción. Podríamos usar una opción no paramétrica.

Ej. 19 — Utilizando el paquete [54, genefilter] se pide diseñar un filtrado no específico para los datos multtest::golub y que seleccione los genes atendiendo a los siguientes criterios:

1. Determinamos para cada gen el coeficiente de variación.
2. Determinamos el percentil 0.90 del los coeficientes de variación para todos los genes.
3. El coeficiente de variación ha de superar el percentil 0.9 de los coeficientes de variación observados.
4. Repetimos los puntos anteriores sustituyendo el coeficiente de variación por el nivel de expresión medio.
5. Conservamos los genes que verifican los dos criterios de selección.

Ej. 20 — Realizar una selección no específica de los datos ALL utilizando el paquete [54, genefilter]. Los criterios de selección son los siguientes:

1. La mediana de los niveles de expresión del gen ha de superar el percentil 0.9 de las medianas observadas para todos los genes.
2. El rango intercuartílico de los niveles de expresión del gen ha de superar el percentil 0.9 de los rangos intercuartílicos observados para todos los genes.

Capítulo 8

Comparaciones múltiples

8.1 Introducción

Empezamos con algunos comentarios bibliográficos. Esta sección se basa fundamentalmente en [39]. Sin embargo, un artículo a consultar es [112].

Supongamos que nuestras muestras corresponden a dos grupos que de un modo genérico podemos llamar casos y controles. Nuestro interés no está en evaluar si uno o dos genes concretos se expresan de un modo distinto entre las dos condiciones consideradas. Se pretende una visión global. Pretendemos responder a una pregunta como: ¿qué genes se expresan de un modo distinto (o diferencial en la jerga habitual en esta literatura) en los dos grupos que consideramos? Tenemos miles de genes: ¿cuáles de ellos tienen realmente una expresión diferencial? Y, lo importante, pretendemos responder la pregunta de modo que controlemos, de algún modo, las veces que admitimos una expresión diferencial cuando no la tiene. ¿Qué contrastes de hipótesis estamos evaluando? Si numeramos los genes con $i = 1, \ldots, N$ entonces para el i-ésimo gen estamos considerando el contraste de hipótesis siguiente:

- H_i: El gen i **no tiene** una expresión diferencial entre las condiciones consideradas.
- K_i: El gen i **tiene** una expresión diferencial entre las condiciones consideradas.

El contraste anterior se puede reformular como

- H_i: La expresión del gen i no tiene asociación con la condición.
- K_i: La expresión del gen i tiene asociación con la condición.

[1]

Estos contrastes nos lo planteamos para cada uno de los N genes que evaluamos. Denotemos por $G = \{1, \ldots, N\}$ el conjunto de hipótesis nulas que estamos evaluando. El número de hipótesis que vamos a contrastar es conocido a priori ya que corresponde con el número de

[1] De este modo, creo que englobamos de un modo más natural el caso en que consideramos asociada a las muestras una covariable que no sea necesariamente un factor experimental con dos niveles.

Hipótesis nula	No rechazadas	Rechazadas	Total
Verdadera	U	V	N_0
Falsa	T	S	$N - N_0 = N_1$
Total	N - R	R	N

Tabla 8.1: Errores tipo I y II en contraste de múltiples hipótesis

genes que estamos evaluando. Denotamos por G_0 (con $G_0 \subset G$) las hipótesis nulas que son ciertas. Denotamos por $N_0 = |G_0|$ el cardinal del conjunto G_0. Notemos que el conjunto G_0 es desconocido para nosotros. No sabemos ni cuántas ni cuáles son las hipótesis nulas ciertas. Esto supondría que conocemos qué genes se expresan diferencialmente y esto es precisamente nuestro objetivo. La situación en la en que nos encontramos viene descrita en la tabla 8.1 (de [13]).

En esta tabla conocemos N, esto es, el número de contrastes de hipótesis. Una vez hemos realizado todos los contrastes y tomado una decisión sobre si rechazamos o no cada hipótesis nula podemos **observar** R que nos indica cuántas hipótesis nulas hemos rechazado. Obviamente R es una variable aleatoria. Distintos datos nos darán distintos valores de R. Los valores de S, T, U, V son también aleatorios. Sin embargo, estas variables no son observables. La variable aleatoria V nos está dando el número (desconocido) de falsos positivos o errores tipo I (no hay expresión diferencial pero decidimos que la hay de un modo erróneo) mientras que T nos da el número de falsos negativos o error tipo II (genes que se expresan diferencialmente pero no admitimos que no lo hacen). Ambas variables indican error y son importantes.

Cuando realizamos un solo contraste el procedimiento consiste en fijar una cota al error tipo I, el nivel de significación, con un valor determinado que solemos denotar por α. Supongamos que denotamos por p_i el p-valor asociado al i-ésimo contraste. Entonces si aplicamos la regla de rechazar H_i cuando $p_i \leq \alpha$ y en otro caso aceptarla (o no rechazarla como se prefiera). Con este procedimiento sabemos que tenemos controlado el error tipo I **para el contraste i-ésimo** a un nivel α. Pero, y esto es fundamental, solamente para el ese contraste. No sabemos nada de lo que ocurre **simultáneamente** para todos los contrastes. Supongamos que tenemos un estadístico T_i que utilizamos para contrastar H_i y supongamos además que rechazamos la hipótesis nula para valores grandes de $|T_i|$[2] En este caso tendremos un valor c_α tal que rechazamos la hipótesis H_i cuando $\{|T_i| > c_\alpha\}$ y se tendrá que

$$P(|T_i| > c_\alpha | H_i) = \alpha.$$

Esto es, la probabilidad de rechazar cuando es cierta la hipótesis nula es α. O la probabilidad de no rechazar cuando es cierta la hipótesis nula H_i será de $1 - \alpha$, es decir,

$$P(|T_i| \leq c_\alpha | H_i) = 1 - P(|T_i| > c_\alpha | H_i) = 1 - \alpha.$$

Cuando realizamos simultáneamente muchos contrastes nos podemos equivocar rechazando la hipótesis nula cuando no lo es en más de una ocasión. De hecho, el número de hipótesis nulas falsamente

[2]Una situación así la tenemos cuando observamos las muestras bajo dos condiciones distintas y utilizamos un test de la t de comparación de medias.

rechazadas es una variable aleatoria que denotamos por V y nos da el número de veces que rechazamos la hipótesis nula erróneamente. Nuestro interés es que la variable aleatoria V tome valores pequeños con probabilidades altas.

Las distintas medidas de error que se utilizan esencialmente cuantifican valores asociados a las variables que hemos definido en la tabla 8.1. Se habla de *tasas de error tipo I* abandonando la expresión de error tipo I utilizada con un solo test. Son extensiones del único error tipo I que se nos plantea cuando realizamos un solo contraste.

Tasa de error por comparación [3] Por $E(V)$ denotamos la media o valor medio de la variable V. Se define la tasa de error por comparación como el cociente entre la media de V y el número de contrastes.

$$PCER = \frac{E(V)}{N},$$

es la proporción esperada de errores tipo I entre las m decisiones que tomamos.

FWER: Tasa de error global [4]

Esta medida de error es la que tiene una mayor tradición en Estadística. Quizás es la forma natural de pensar: no quiero equivocarme nunca rechazando una hipótesis nula cuando es cierta. La definimos como

$$FWER = P(V > 0) = P(V \geq 1).$$

Sin duda es la medida ideal. Ningún error. Esto tiene un pago. Es un criterio muy exigente si tenemos un número de hipótesis N muy grande. Notemos que nos fijamos en cometer al menos un error cuando con frecuencia tendremos decenas de miles de contrastes. Esto puede parecer algo bueno pero también estaremos construyendo tests que procurarán no rechazar una hipótesis nula salvo que tengan una evidencia contra ella muy grande. Procedimientos para contrastar miles de hipótesis nula y que hagan pequeña esta tasa de error tenderán a ser muy conservadores. Como (mala) contrapartida muchos genes que tendrán una expresión diferencial realmente no serán detectados. O dicho de un modo más técnico perderemos potencia en los contrastes donde la potencia es la probabilidad de rechazar H_i cuando realmente es falsa.

FDR: Tasa de falsamente rechazados [5] Se propuso en [13]. De alguna manera se vió que la tasa FWER era demasiado exigente y se trataba de conseguir procedimientos más potentes sin que por ello dejaramos de tener una tasa de error razonable controlada. Definimos la variable aleatoria $Q = V/R$ si $R > 0$ y $Q = 0$ en otro caso. Esta variable simplemente nos da la proporción de test que rechazamos erróneamente. Rechazamos R de los cuales V no debieran de ser rechazados, por tanto, nuestra proporción de hipótesis nulas falsamente rechazadas es Q. Tenemos el problema de si no rechazamos ninguna hipótesis. En este caso

[3] Per-comparison error rate.
[4] Familywise error rate.
[5] False discovery rate.

tendremos 0/0. Lo que se hace es definir el cociente como cero. Otra vez, Q es algo aleatorio, es una proporción aleatoria. ¿Qué queremos controlar de este valor aleatorio? La conocida como tasa de falsamente rechazados o *false discovery rate* denotada como FDR y que se define como

$$FDR = E(Q) = E\left(\frac{V}{R}|R>0\right)P(R>0)+0\times P(R=0) = E\left(\frac{V}{R}|R>0\right)P(R>0),$$

que nos da la proporción esperada de hipótesis erróneamente rechazadas entre aquellas hipótesis que hemos rechazado. A la tasa FDR la hemos llamada tasa de falsamente rechazados. Hay una cierta confusión con la tasa de falsos positivos. Esta tasa sería la proporción de hipótesis nulas que son ciertas y que son rechazadas. Esto es, sería el valor medio de la variable V/N_0. Por ejemplo, si tenemos una FDR de 5% entonces el 5% de las hipótesis nulas que rechazamos son realmente ciertas. Si tenemos una tasa de falsos positivos del 5% entonces una media del 5% de las hipótesis nulas ciertas son erróneamente rechazadas. Si tenemos p-valores p_i y aplicamos simplemente la regla de rechazar H_i cuando $p_i \leq \alpha$ entonces tenemos una tasa de falsos positivos α.

pFDR: Tasa de falsamente rechazados modificada [6] Es una modificación de la anterior. Se define como

$$pFDR = E\left(\frac{V}{R}|R>0\right).$$

Cuando tenemos un gran número de hipótesis nulas N entonces la probabilidad de que rechacemos al menos una es prácticamente segura, es decir, $P(R>0) \approx 1$ y por lo tanto ambas tasas de error son también prácticamente iguales $FDR \approx pFDR$. Definimos la tasa pFDR porque está muy relacionada con el concepto de q-valor que definimos más adelante.

En lo que sigue vamos a considerar distintos procedimientos para contrastar muchas hipótesis. Un procedimiento concreto se dice que controla una tasa de error tipo I al nivel α cuando su tasa de error es menor o igual que α cuando aplicamos el procedimiento para producir una lista de R hipótesis nulas rechazadas o de genes declarados como significativos. La tasa de error que utilizaremos fundamentalmente en lo que sigue será la tasa FDR.

8.2 Control fuerte y control débil del error

Cuando hemos definido las tasas de error hemos considerado el comportamiento aleatorio de V y R (conjunto y marginal de hecho). Ese comportamiento aleatorio depende de qué hipótesis nulas son ciertas y qué hipótesis son falsas. Por ejemplo, cuando hemos definido

$$FWER = P(V \geq 1),$$

realmente la probabilidad depende de qué hipótesis nulas son ciertas y qué hipótesis nulas son falsas. Lo correcto hubiera sido escribir:

$$FWER = P(V \geq 1 \mid \cap_{i\in M_0} H_i),$$

[6] Positive false discovery rate.

es decir, la probabilidad de rechazar al menos una hipótesis nula equivocadamente condicionando a que se verifican las hipótesis de M_0.[7] Un detalle importante es que cuándo controlamos una tasa de error tipo I es saber qué hipótesis nulas asumimos que son ciertas. Un par de definiciones que son importantes.

Definición 8.1 (Control fuerte) *Se dice que un procedimiento de contraste de múltiples hipótesis realiza un control fuerte de la tasa de error tipo I elegida cuando dicha tasa de error es menor o igual que el límite que especifiquemos para cualquier combinación de hipótesis ciertas y falsas, en otras palabras, para cualquier subconjunto $M_0 \subset \{1, \ldots, m\}$ de hipótesis ciertas (el resto las suponemos falsas).*

Definición 8.2 (Control débil) *Se dice que un procedimiento de contraste de múltiples hipótesis realiza un control débil de la tasa de error tipo I elegida cuando dicha tasa de error es menor o igual que el límite que especifiquemos asumiendo que todas las hipótesis nulas son ciertas. Asumimos pues que se verifica la hipótesis nula*

$$H_0^C = \cap_{i=1}^N H_i.$$

En el control débil asumimos algo que no tiene muchos visos de realidad. Estamos asumiendo que **todas las hipótesis nulas son ciertas**. Si pensamos en nuestro problema estamos diciendo que ningún gen tiene una expresión diferencial entre condiciones. No sé pero mal lo tenemos si es así. Parece poco realista asumir esto. Lo natural es asumir que algunas hipótesis nulas son ciertas y otras falsas. Es decir, que existe un conjunto N_0 que nos da las que son ciertas y aquellos que no están en N_0 son falsas. ¿Y quién es N_0? Ese es nuestro problema precisamente. En el control fuerte nos preparamos *para cualquier* N_0 sea el que sea. En lo que sigue cuando pongamos probabilidades referidas a V (o cualquier otra variable) siempre han de entenderse como probabilidades condicionadas a la realidad (desconocida para nosotros), es decir, probabilidades condicionadas a $\cap_{i \in N_0} H_i$ siendo N_0 el conjunto de hipótesis nulas ciertas (en nuestro caso, los genes que están en N_0 no se expresan de un modo diferencial y sí lo hacen los que no están en N_0).

8.3 Relación entre las tasas de error tipo I

En general se verifican las siguientes desigualdades:

$$PCER \leq FDR \leq FWER.$$

Esto significa que si un procedimiento controla la FWER entonces tenderá a rechazar menos hipótesis nulas, tenderá a ser más conservador por tanto y, posiblemente, nos estaremos perdiendo genes que tienen expresión diferencial y no lo apreciamos.

La aproximación clásica al problema de las contrastes múltiples se basa en el control fuerte de la FWER. Una aproximación más reciente propuesta en [13] consiste en controlar la FDR en un sentido débil.

[7] Un detalle, no conocemos M_0.

8.4 p valores y p valores ajustados

Para cada contraste H_i tendremos un p-valor p_i. En el test de la t para comparar medias tenemos

$$p_i = P(|T_i| \geq t_i | H_i).$$

donde t_i es el i-ésimo estadístico observado. A los p-valores p_i los llamaremos p-valores originales. Podemos contrastar el contraste H_i con un nivel de significación α_i rechazando la hipótesis nula H_i si $p_i \leq \alpha_i$ y no rechazando en otro caso. Cuando consideramos simultáneamente todos los tests los valores α_i serán distintos y por ello tendríamos que ir comprobando si la desigualdad $p_i \leq \alpha_i$ se verifica o no. La idea del p-valor ajustado es transformar el p-valor p_i en otro valor $\tilde{p}_i$, el i-ésimo p-valor ajustado, de modo que sean equivalentes: $p_i \leq \alpha_i$ y $\tilde{p}_i \leq \alpha$ siendo α el valor que especificamos para controlar alguna de las tasas de error tipo I.

8.5 Métodos que controlan la FWER

Veremos procedimientos que actúan en un solo paso y procedimientos por pasos bien de bajada o de subida.

8.5.1 Los métodos en un solo paso

Veamos el ejemplo más conocido, es el método de Bonferroni. Supongamos que tenemos el p-valor p_i asociado al contraste H_i. Si solamente estuvieramos contrastando la hipótesis H_i entonces rechazamos con un error tipo I α si $p_i < \alpha$ y aceptamos o no rechazamos en otro caso. La modificación de Bonferroni es simple. Ahora tenemos en cuenta que hay m contrastes y rechazamos H_i si

$$p_i \leq \frac{\alpha}{N} \tag{8.1}$$

o, equivalentemente, si

$$Np_i \leq \alpha. \tag{8.2}$$

Si tenemos en cuenta que $0 < \alpha \leq 1$ entonces la desigualdad 8.2 es equivalente a que

$$\min\{Np_i, 1\} \leq \alpha. \tag{8.3}$$

De este modo el p-valor ajustado será $\tilde{p}_i = \min\{Np_i, 1\}$. [8]

[8] El método de Bonferroni controla la FWER de un modo débil. Veámoslo. Denotemos por E_i el suceso consistente en que no rechazamos H_i o, si se prefiere, aceptamos H_i. Se busca que $P(\cap_{i=1}^N E_i | \cap_{i=1}^N H_i) = 1 - \alpha$. Las probabilidades que siguen se entienden que son condicionadas a que todas las hipótesis nulas son simultáneamente ciertas. Tenemos

$$P(\cap_{i=1}^N E_i) = 1 - P(\cup_{i=1}^N E_i^c) \leq 1 - \sum_{i=1}^N P(E_i^c). \tag{8.4}$$

Pero $P(E_i^c)$ es la probabilidad de rechazar H_i siendo cierta $\cap_{i=1}^N H_i$. Supongamos que consideramos una región crítica de tamaño α/m, es decir, $P(E_i^c) \leq \alpha/m$. Teniendo en cuenta 8.4 se sigue

$$P(\cap_{i=1}^N E_i) \leq 1 - \sum_{i=1}^N P(E_i^c) \leq 1 - m\frac{\alpha}{N}. \tag{8.5}$$

Método de Bonferroni Este método nos da un control fuerte de la FWER al nivel α. En este método rechazamos la hipótesis nula H_i si el correspondiente p-valor sin ajustar es menor o igual a $\frac{\alpha}{N}$. El p-valor ajustado sería:

$$\tilde{p}_i = \min\{Np_i, 1\}$$

Método de Sĭdák Nos permite un control débil del FWER. Los p-valores ajustados son

$$\tilde{p}_i = 1 - (1 - p_i)^N.$$

Asume la independencia de los p valores sin ajustar, en definitiva, asume la independencia de los tests. En nuestro contexto no es muy asumible esta hipótesis pues los genes tienen expresiones relacionadas, no independientes.

Método de minP en un solo paso Los p-valores ajustados son

$$\tilde{p}_i = P\left(\min_{1\leq l\leq N} P_l \leq p_i \mid H_o^C\right),$$

siendo H_o^C la hipótesis nula completa (la intersección de todas las hipótesis nulas) y P_l la variable aleatoria que nos da el p-valor aleatorio de la l-ésima hipótesis.

maxT en un solo paso Podemos considerar como una opción alternativa como el máximo de los valores de los estadísticos observados. Es el maxT en un solo paso donde los p-valores ajustados son

$$\tilde{p}_i = P\left(\max_{1\leq l\leq N} \mid T_l \mid \geq \mid t_i \mid \mid H_o^C\right),$$

Los procedimientos basados en maxT o minP nos dan un control débil de FWER.

En general, los métodos de un solo paso son simples de implementar pero tiende a ser conservadores, esto es, tienden a no rechazar la hipótesis nula. No son pues muy potentes en el sentido de detectar genes con expresión diferencial. Los métodos por pasos son más potentes.

8.5.2 Los métodos por pasos de bajada

Los p-valores originales son $p_1, \ldots, p_m$. Los ordenamos de menor a mayor:

$$p_{r_1} \leq p_{r_2} \leq \ldots \leq p_{r_N}.$$

Tendremos las correspondientes hipótesis nulas $H_{r_1} \leq H_{r_2} \leq \ldots \leq H_{r_N}$

Método de Holm Definimos

$$i^* = \min\{i : p_{r_i} > \frac{\alpha}{m - i + 1}\}$$

y rechazamos las hipótesis nulas H_{r_i} para $i = 1, \ldots, i^* - 1$. Si no existe i^* entonces rechazamos todas las hipótesis nulas. Si consideramos los p-valores ajustados entonces vienen dados por

$$\tilde{p}_{r_i} = \max_{k=1,\ldots,i}\{\min\{m - k + 1)p_{r_k}, 1\}\}$$

El método de Holm es menos conservador que el método de Bonferroni.

Método de Sĭdák por pasos de bajada Definimos los p-valores ajustados como:

$$\tilde{p}_{r_i} = \max_{k=1,\ldots,i}\{1-(1-p_{r_k})^{m-k+1}\}.$$

min P por pasos de bajada Se definen los p-valores ajustados como:

$$\tilde{p}_{r_i} = \max_{k=1,\ldots,i}\left\{P\left(\min_{l\in\{r_k,\ldots,r_m\}} P_l \leq p_{r_k} \mid H_o^C\right)\right\}$$

max T por pasos de bajada Los p-valores ajustados son:

$$\tilde{p}_{r_i} = \max_{k=1,\ldots,i}\left\{P\left(\max_{l\in\{r_k,\ldots,r_m\}} \mid T_j \mid \geq \mid t_{r_k} \mid\mid H_o^C\right)\right\}$$

donde $|t_{r_1}| \geq |t_{r_2}| \geq \ldots \geq |t_{r_N}|$ son los estadísticos ordenados.

8.5.3 Método por pasos de subida

Vamos a ver el método de Hochberg. Se pretende, como en los anteriores, controlar el FWER a un nivel α. Consideramos

$$i^* = \max\{i : p_{r_i} \leq \frac{\alpha}{m-i+1}\}$$

y se rechazan las hipótesis nulas H_{r_i} para $i = 1,\ldots,i^*$. Los p-valores ajustados de Hochberg son

$$\tilde{p}_{r_i} = \min_{k=i,\ldots,N}\big\{\min\{(m-k+1)p_{r_k},1\}\big\}.$$

Este procedimiento es una versión en sentido inverso del procedimiento de Holm.

8.6 Métodos que controlan el FDR

Los métodos de la sección anterior controlan el FWER. En resumen estamos controlando el no cometer ningún error tipo I (admitir un gen con expresión diferencial cuando no la tiene). Esto produce procedimientos conservadores. En [13] se propuso la idea de que cuando contrastamos muchas hipótesis podemos tolerar algunos errores tipo I, siempre que el número de estos errores sea pequeño en relación con el número de hipótesis que se rechazan. Esta es la idea de controlar el FDR. Dos son los procedimientos que vamos a ver para controlar esta tasa.

Benjamini y Hochberg Este procedimiento fue propuesto en [13]. Siendo $p_{r_1} \leq \ldots \leq p_{r_N}$ son los p-valores originales ordenados entonces consideramos

$$i^* = \max\{i : p_{r_i} \leq \frac{i}{N}\alpha\}$$

y rechazamos H_{r_i} para $i = 1,\ldots,i^*$. Si no existe i^* entonces no rechazamos ninguna hipótesis. Los p-valores ajustados se definen como

$$\tilde{p}_{r_i} = \min_{k=i,\ldots,N}\left\{\min\left\{\frac{N}{k}p_{r_k},1\right\}\right\}.$$

Benjamini y Yekutieli Propuesto en [14]. Como en el anterior, $p_{r_1} \leq \ldots \leq p_{r_N}$ son los p-valores originales ordenados. En este caso los p-valores ajustados se definen como

$$\tilde{p}_{r_i} = \min_{k=i,\ldots,N} \left\{ \min \left\{ \frac{m \sum_{j=1}^{N} 1/j}{k} p_{r_k}, 1 \right\} \right\}.$$

8.7 Utilizando genefilter y p.adjust

En esta sección incluimos análisis de expresión diferencial (marginal) utilizando los paquetes [108, multtest] y [54, genefilter]. No vamos a realizar en principio ninguna selección no específica y nos planteamos trabajar con todos los genes.

8.7.1 Cálculo de p-valores ajustados

Calculamos los estadísticos t (test de la t con varianzas distintas) y los p-valores asociados.

Utilizamos los datos tamidata::gse1397.

```
data(gse1397,package ="tamidata")
eset = gse1397; y = pData(gse1397)[,"type"]
```

Aplicamos los t-test's para cada gen.

```
tt = genefilter::rowttests(eset,y)
```

Los p-valores originales los guardamos en `p0`.

```
p0 = tt[,"p.value"]
```

Supongamos que vamos a utilizar el método de Benjamini-Hochberg. Entonces pasamos de los p-valores originales (raw p-values) a los p-valores ajustados con la función stats::p.adjust.

```
p.BH = p.adjust(p0, method = "BH")
```

Podemos incorporar los p-valores ajustados al data.frame original.

```
tt1 = data.frame(tt,p.BH)
```

Podemos utilizar otros procedimientos de ajuste. Para ver las opciones lo mejor es consultar la ayuda de stats::p.adjust.

8.7.2 Genes con expresión diferencial

Lo primero es fijar una tasa de error α. En concreto podemos considerar el valor $\alpha = 0.05$.

```
alpha = 0.05
```

Consideremos los p-valores ajustados utilizando Benjamini-Hochberg manteniendo el orden original de la matriz de expresión. Observad que la segunda columna tiene los p-valores ajustados por el método Benjamini-Hochberg.

```
p1 = p.adjust(p0, "BH")
```

Los genes significativos, esto es, aquellos que tienen un p-valor ajustado menor a la tasa especificada ocupan las siguientes filas de la matriz de expresión.

```
(significativos.BH  = which(p.BH < alpha))

## [1]  346 1170 1853 6303 7889
```

¿Cuántos eran significativos con los p-valores originales?

```
significativos.p0  = which(p0 < alpha)
length(significativos.p0)

## [1] 902
```

8.8 El q-valor

Esta sección se basa en [129]. Supongamos que ordenamos de menor a mayor los p-valores calculados para los distintos contrastes.

$$p_{(1)} \leq \cdots \leq p_{(N)}$$

Cuando hacemos este ordenamiento, como hemos visto en los procedimientos anteriores, declarar significativo uno de estos p-valores supone declarar significativos a todos los que son menores o iguales que él. Los métodos anteriores buscaban un punto de corte para estos p-valores ordenados. Por debajo de este punto de corte admitimos que los genes se expresan diferencialmente (rechazamos al hipótesis nula correspondiente) y por encima aceptamos la hipótesis nula. La idea del q-valor es asociar a cada test un valor numérico que nos diga lo extremo que es su p-valor considerando el resto de p-valores. De un modo intuitivo, si consideramos un test determinado nos da la proporción esperada de falsos positivos en la que incurrimos cuando declaramos significativo ese test.

¿Qué es el q-valor?

Posiblemente el mejor modo de entender qué es el q-valor sea ver un buen procedimiento para estimarlo propuesto en [129] e implementado en el paquete [8, qvalue]. Fijamos un valor t y consideremos que rechazamos H_i cuando $P_i \leq t$.[9] Consideremos los siguientes valores

$$V(t) = |\{P_i : P_i \leq t; H_i \text{ es cierta}; i = 1, \ldots, N\}| \tag{8.6}$$

y

$$R(t) = |\{P_i : P_i \leq t; i = 1, \ldots, N\}| \tag{8.7}$$

Se puede aproximar la pFDR con

$$E\left[\frac{V(t)}{R(t)}\right]$$

[9] Como es habitual denotamos por P_i el p-valor antes de ser observado, esto es, el valor aleatorio antes de tomar los datos.

Tenemos un gran número de contrastes N por ello aproximadamente se tiene que

$$E\left[\frac{V(t)}{R(t)}\right] \approx \frac{EV(t)}{ER(t)}.$$

Podemos estimar $R(t)$ simplemente contando el número de p_i observados que son menores o iguales a t,

$$\hat{R}(t) = |\{p_i : p_i \leq t\}|$$

Para estimar $V(t)$ hemos de tener en cuenta que si la hipótesis nula H_i es cierta entonces el p-valor aleatorio P_i se distribuye uniforme en el intervalo $[0, 1]$ y por tanto la probabilidad de que sea menor o igual a t es precisamente t,

$$P(P_i \leq t | H_i) = t.$$

Por tanto

$$EV(t) = N_0 t.$$

Pero no conocemos N_0 número de hipótesis nulas ciertas. Lo que vamos a hacer es estimar la proporción de hipótesis nulas ciertas $\pi_0 = N_0/N$ (notemos que el total de hipótesis, N, sí es conocido). Bajo H_i, el p-valor aleatorio P_i es uniforme en $[0, 1]$, por tanto los p-valores correspondientes a hipótesis nulas ciertas se distribuyen sobre todo el intervalo $[0, 1]$, sin embargo, aquellos p-valores muy pequeños, muy próximos a cero, es más probable que correspondan a hipótesis nulas falsas. Fijamos un valor $\lambda \in [0, 1]$, un estimador razonable para π_0 es

$$\hat{\pi}_0 = \frac{|\{p_i : p_i > \lambda; i = 1, \ldots, N\}|}{N(1 - \lambda)}.$$

Podemos pues estimar pFDR con

$$\widehat{pFDR} = \frac{\hat{\pi}_0 N t}{|\{p_i : p_i \leq t\}|}.$$

El q-valor asociado a un contraste sería el mínimo valor de pFDR que se alcanza cuando el contraste es rechazado. Es decir, el q-valor asociado al test i-ésimo sería

$$q(p_i) = \min_{t \geq p_i} pFDR(t)$$

y su estimador sería

$$\hat{q}(p_i) = \min_{t \geq p_i} \widehat{pFDR}(t).$$

El estimador $\hat{\pi}_0$ depende de un valor λ. De hecho el procedimiento exacto no utiliza un valor λ concreto sino que estima utilizando distintos valores y luego hace un ajuste tomando como estimador el valor estimado a partir del ajuste en 1.

El algoritmo exacto es el siguiente:

1. Sean $p_{(1)} \leq \ldots \leq p_{(N)}$ los p-valores ordenados.
2. Para un rango de valores de λ, por ejemplo, λ de 0 a 0.95 con incrementos de 0.01 calculamos

$$\hat{\pi}_0(\lambda) = \frac{|\{p_i : p_i > \lambda; i = 1, \ldots, N\}|}{N(1 - \lambda)}.$$

3. Determinamos $\hat{f}$ el spline natural cúbico con 3 grados de libertad a partir de los valores $\hat{\pi}_0(\lambda)$.

4. Fijamos como estimador de π_0 el valor

$$\hat{\pi}_0 = \hat{f}(1).$$

5. Calculamos

$$\hat{q}(p_{(m)}) = \min_{t \geq p_{(N)}} \frac{\hat{\pi}_0 N t}{|\{p_i : p_i \leq t\}|} = \hat{\pi}_0 p_{(N)}.$$

6. Para $i = N-1, \ldots, 1$ calculamos

$$\hat{q}(p_{(i)}) = \min_{t \geq p_{(i)}} \frac{\hat{\pi}_0 N t}{|\{p_i : p_i \leq t\}|} = \min\{\frac{\hat{\pi}_0 N p_{(i)}}{i}, \hat{q}(p_{(i+1)})\}.$$

7. El q-valor estimado para el i-ésimo contraste más significativo es $\hat{q}(p_{(i)})$.

Calculando el q-valor

El método de estimación que hemos visto en § 8.8 está implementado en el paquete [8, qvalue].[10]

Utilizamos los datos tamidata::gse1397.

```
pacman::p_load("Biobase")
data(gse1397,package="tamidata")
tt = genefilter::rowttests(gse1397,pData(gse1397)[,"type"])
pvalue = tt$p.value
```

Utilizamos el paquete [8, qvalue].

```
pacman::p_load(qvalue)
qobj = qvalue(pvalue)
```

Podemos ver una descripción gráfica con

```
plot(qobj)
```

que aparece en la figura 8.1. En concreto tenemos:

Arriba izquierda: Los distintos valores estimados de π_0, proporción de hipótesis nulas ciertas, cuando variamos el valor de λ. El valor estimado por el método antes propuesto nos da un 96.3% de genes sin expresión diferencial.

Arriba derecha: La transformación que nos lleva de los p-valores originales (ordenados de menor a mayor) a los q-valores.

Abajo izquierda: Para distintos puntos de corte para los q-valores el número de genes que declaramos con expresión diferencial significativa.

Abajo derecha: En abscisas el número de test significativos y en ordenadas el número de falsos positivos esperado.

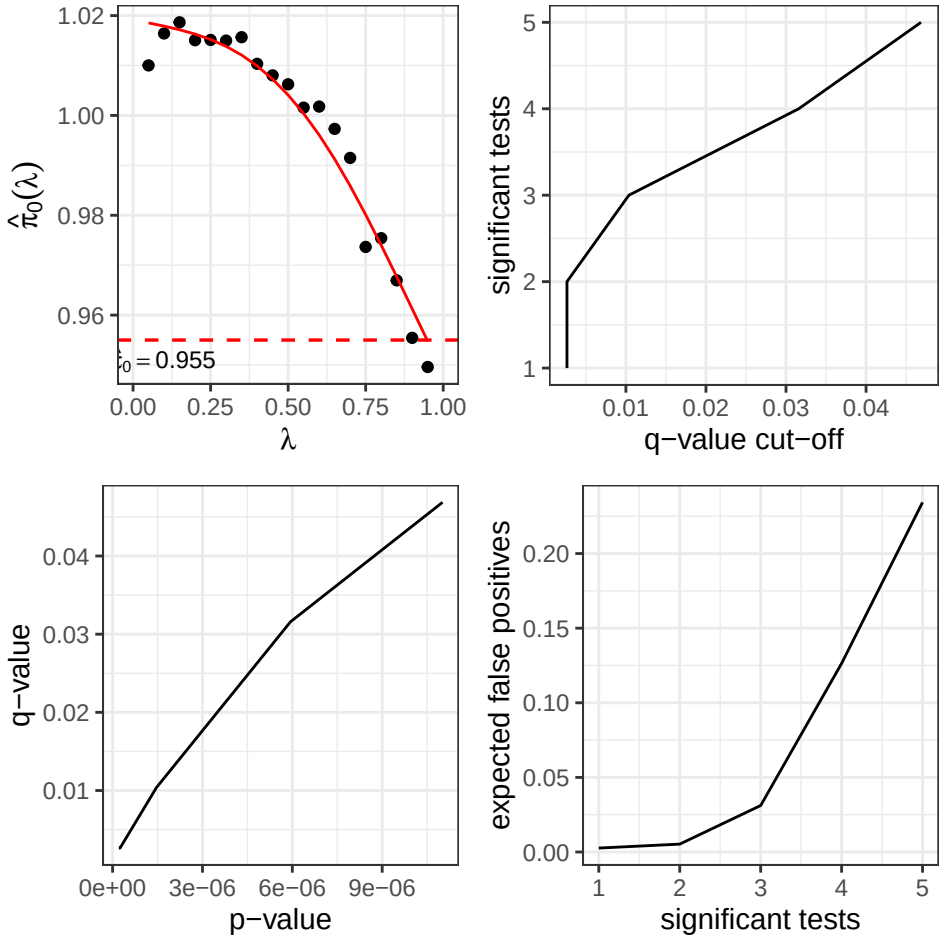


Figura 8.1: Distintos gráficos asociados a un objeto qvalue::qvalue.

Si simplemente queremos los q-valores los obtenemos con

```
q.value = qvalue(pvalue)$qvalues
```

8.9 Ejercicios

Ej. 21 — Utilizando los datos tamidata::gse20986 se pide:

1. Seleccionar las muestras correspondientes a iris y huvec.
2. Aplicamos, a cada gen, un test de la t (asumiendo una misma varianza). Obtener los t-estadísticos y los p-valores correspondientes. [149]
3. Utilizando el método de corrección de Benjamini-Hochberg obtener los p-valores ajustados.[150]
4. Si utilizamos un valor de FDR igual a 0.05: ¿qué genes declaramos como significativos?
5. Repetir los apartados anteriores comparando los muestras correspondientes a retina y huvec.
6. Repetir los apartados anteriores comparando las muestras correspondientes a coroides y huvec.
7. Utilizando la función base::intersect y la función Biobase::featureNames determinar los genes comunes a cada par de comparaciones y los comunes a las tres comparaciones.

[149] Utilizad genefilter::rowttests.

[150] Utilizad [108, multtest].

Ej. 22 — Vamos a utilizar los datos tamidata::gse1397. Vamos a realizar un estudio de expresión diferencial marginal comparando las personas con y sin la trisomía del cromosoma 21.

1. Aplicamos, a cada gen, un test de la t (asumiendo una misma varianza). Obtener los t-estadísticos y los p-valores correspondientes.
2. Si aplicamos la regla de decisión consistente en admitir una expresión diferencial cuando el p-valor es menor que $\alpha = 0.05$: ¿cuántos y qué genes son declarados significativos?
3. Utilizando los métodos de corrección de Bonferroni y de Benjamini-Hochberg obtener los p-valores ajustados para cada uno de los procedimientos.
4. Representar, en un mismo dibujo, los p-valores originales y los ajustados obtenidos en el paso anterior. Utilizad tipos de línea y colores diferentes para cada conjunto de p-valores.
5. Si utilizamos un valor de FDR igual a 0.05: ¿qué genes declaramos como significativos utilizando la corrección de Benjamini-Hochberg? ¿Y utilizando la corrección de Bonferroni?
6. ¿Todos los significativos según Bonferroni lo son con Benjamini-Hochberg? Comparar los grupos de genes significativos utilizando las funciones base::intersect y Biobase::featureNames.

[10] También se pueden bajar programas que no requieren R para distintos sistemas operativos. Se puede encontrar en http://genomics.princeton.edu/storeylab/qvalue/.

Capítulo 9

Expresión diferencial con microarrays

9.1 Método SAM

Es la abreviatura de **S**ignificance **A**nalysis of **M**icroarrays. Fue propuesto en [136].[1]

9.1.1 El método

El conjunto de muestras, como es habitual, lo denotaremos como por $\{1, \ldots, n\}$. Empezamos con el caso en que pretendemos comparar dos grupos. Esta información la podemos tener con un vector $y = (y_1, \ldots, y_n)$ donde y_j vale 1 si la muestra j está en el primer grupo y vale 2 si la muestra en esa columna está en el segundo grupo. Para un gen dado las expresiones del grupo 1 (o del grupo 2) corresponden con los valores donde $y_j = 1$ (respectivamente $y_j = 2$). Consideramos $J_g = \{j : y_j = g\}$ con $g = 1, 2$. Tendremos J_1 y J_2 donde J_g son las muestras del grupo g con g= 1,2.[2] Denotamos por n_1 y n_2 los cardinales de J_1 y J_2. Podemos denotar

$$\bar{x}_{ig} = \bar{x}_{i,J_g} = \frac{\sum_{j \in J_g} x_{ij}}{n_g}$$

siendo n_g el número de muestras en J_g. Para el i-ésimo gen consideramos la *diferencia relativa* dada por

$$d_i = \frac{\bar{x}_{i1} - \bar{x}_{i2}}{s_i + s_0} \tag{9.1}$$

donde

$$s_i = \sqrt{a \sum_{g=1}^{2} \sum_{j \in J_g} (x_{ij} - \bar{x}_{i,g})^2} \tag{9.2}$$

[1] Tiene un gran interés consultar la dirección http://www-stat.stanford.edu/~tibs/SAM/. De hecho, no se puede entender el método tal cual se aplica hoy a partir de la referencia original. Ha habido una gran evolución posterior del procedimiento incluyendo el manejo de datos de secuencia. Existe una versión comercial para su uso con Excel. En la parte de los detalles técnicos del manual de la versión comercial se puede encontrar la explicación detallada del método y, por lo tanto, de las salidas que realizan las funciones. Este manual técnico lo podemos conseguir en http://www-stat.stanford.edu/~tibs/SAM/sam.pdf.

[2] Notemos que J_1 y J_2 son una partición de $\{1, \ldots, n\}$ de modo que $J_1 \cup J_2 = \{1, \ldots, n\}$ y son disjuntos $J_1 \cap J_2 = \emptyset$.

y

$$a = \frac{1/n_1 + 1/n_2}{n_1 + n_2}$$

Observemos que el estadístico d_i es el estadístico del contraste de igualdad de medias o simplemente el t-estadístico al que le modificamos el error estándar. ¿Para qué se hace esta modificación? Si el valor s_i es muy pequeño entonces el valor d_i es muy grande. En valores de expresión pequeños esto puede pasar. Los distintos valores d_i no son comparables para los distintos genes (la expresión que utilizan los propios autores es que no son intercambiables). Por ello se le suma un valor s_0 que *estabiliza* la varianza (y nos permite comparar los t-valores entre si). Este valor hay que *estimarlo* o calcularlo a partir de los propios datos (§ 9.1.2). Veamos el procedimiento enumerando los pasos.

1. Calculamos los estadísticos d_i según la ecuación 9.1.

2. Ordenamos de menor a mayor los d_i's observados: $d_{(1)} \leq \ldots \leq d_{(N)}$ de modo que $d_{(i)}$ es el i-ésimo menor.

3. Realizamos B permutaciones aleatorias del vector y que nos indica el grupo de la columna. Para cada permutación calculamos los nuevos valores d_i y los denotamos por $d_i(b)$. Ordenamos, como antes, estos nuevos valores de menor a mayor: $d_{(1)}(b) \leq \ldots \leq d_{(N)}(b)$.

4. Calculamos, para las B permutaciones, el valor medio de estos estadísticos ordenados, es decir, calculamos

$$\bar{d}_{(i)} = \sum_{b=1}^{B} \frac{d_{(i)}(b)}{B}.$$

   Bajo la hipótesis nula de que no hay diferencias entre los dos grupos los valores de $d_{(i)}$ y de $\bar{d}_{(i)}$ debieran de ser *similares*.

5. Representamos gráficamente $d_{(i)}$ frente a $\bar{d}_{(i)}$.

6. Para un valor fijo positivo Δ, empezando en el origen y moviéndonos a la derecha determinamos el primer $i = i_1$ tal que $d_{(i)} - \bar{d}_{(i)} > \Delta$. Todos los genes a la derecha de i_1 son llamados *significativamente positivos* (estos genes verifican la desigualdad anterior). Del mismo modo, empezando en el origen y moviéndonos a la izquierda determinamos el primer $i = i_2$ tal que $\bar{d}_{(i)} - d_{(i)} > \Delta$. Todos los genes con la diferencia anterior más a la izquierda de i_2 los llamamos *significativamente negativos*. Para cada Δ definimos un punto de corte superior como

$$u(\Delta) = \min\{d_i : i \text{ es significativamente positivo}\}$$

   y

$$l(\Delta) = \max\{d_i : i \text{ es significativamente negativo}\}$$

7. Para distintos valores de Δ, calculamos:

   - El número total de genes significativos (como hemos visto en el paso anterior).

- En la b-ésima permutación aleatoria hemos obtenido $d_i(b)$ con $i = 1, \ldots, N$. Calculamos el número de genes *falsamente llamados* en la aleatorización b-ésima como

$$c_b = |\{i : d_i(b) > u(\Delta) \text{ o } d_i(b) < l(\Delta)\}$$

Tendremos los valores c_b con $b = 1, \ldots, B$. Notemos que los valores $d_i(b)$ han sido calculados bajo la hipótesis de no asociación, es cierta la hipótesis nula para cada gen. En consecuencia debieran de estar en el intervalo $[l(\Delta), u(\Delta)]$. Calculamos la mediana y el percentil de orden 0.90 de los valores c_b con $b = 1, \ldots, B$ y los denotamos como $q_{0.5}(c)$ y $q_{0.90}(c)$.

8. Vamos a estimar π_0, la proporción de genes sin expresión diferencial, de genes que no se asocian con la covariable y. En definitiva cuál es la proporción de hipótesis nulas ciertas.

   (a) Calculamos los percentiles de orden 0.25 y 0.75, $q_{0.25}$ y $q_{0.75}$, de todos los valores $d_i(b)$ que hemos obtenido realizando las permutaciones: $\{d_i(s); i = 1, \ldots, N; s = 1, \ldots, B\}$ (tenemos un total de $N \times B$ valores).

   (b) Calculamos

$$\hat{\pi}_0 = \frac{|\{d_i : d_i \in (q_{0.25}, q_{0.75})\}|}{N/2}$$

   donde $\{d_1, \ldots, d_N\}$ son los d valores originales.

   (c) Truncamos el valor de $\hat{\pi}_0$ en 1, es decir, consideramos

$$\hat{\pi}_0 = \min\{\hat{\pi}_0, 1\}.$$

   [3]

9. Multiplicamos la mediana, $q_{0.5}(c)$, y el percentil 0.9, $q_{0.90}(c)$, determinados en el paso 7 por $\hat{\pi}_0$.

10. Elegimos un valor de Δ y determinamos el conjunto de genes significativos.

11. La tasa de falsos positivos FDR es estimada como el cociente entre la mediana, $q_{0.5}(c)$, (o el percentil 0.90, $q_{0.90}(c)$) del número de genes falsamente llamados y el número de genes declarados significativos.

12. El **q-valor** (que aparece en la salida de [133, samr]) es la tasa pFDR cuando la lista de genes significativos está compuesta por este gen y todos los genes que son más significativos que este. Se calcula buscando cuál es el valor más pequeño de Δ para el cual este gen es declarado como significativo. Si este valor es $\hat{\Delta}$ entonces la pFDR asociada a este valor es el q-valor asociado al gen.

[3] En la opción en que estamos comparando varios grupos que vemos más adelante los valores de d son todos positivos y se trabaja con los percentiles 0 y 0.5.

9.1.2 Cálculo de s_0

La selección de s_0 se hace del siguiente modo:

1. Consideramos $\boldsymbol{s} = (s_1, \ldots, s_N)$ las desviaciones estándar muestrales.
2. Sea $q_\alpha(\boldsymbol{s})$ el percentil de orden α de los valores s_i. Definimos
$$d_i^{(\alpha)} = \frac{\bar{x}_{i1} - \bar{x}_{i2}}{s_i + q_\alpha(\boldsymbol{s})}.$$
3. Consideramos los percentiles $q_{j/100}(\boldsymbol{s})$ con $j = 1, \ldots, 100$.
4. Para cada $\alpha \in \{0, 0.1, \ldots, 1\}$:
   (a) Se calcula
   $$v_j = \frac{1}{0.64} MAD(\{d_i^{(\alpha)} : s_i \in [q_{j/100}(\boldsymbol{s}), q_{(j+1)/100}(\boldsymbol{s})]\})$$
   para $j = 1, \ldots, 100$ donde MAD es la mediana de las desviaciones absolutas respecto de la mediana.
   (b) Calculamos el valor $CV(\alpha)$, el coeficiente de variación de los valores v_j.[151]
   (c) Se elige como valor de α: $\hat{\alpha}$ que minimiza $CV(\alpha)$.
   (d) Finalmente tomamos para s_0: $\hat{s}_0 = q_{\hat{\alpha}}(\boldsymbol{s})$.

[151] El coeficiente de variación es el cociente de la media y la desviación estándar.

9.1.3 SAM con samr

Utilizamos los datos golub. Cargamos el paquete [133, samr].

```
pacman::p_load("samr")
```

Realizamos el análisis. Tenemos dos grupos y en el tipo de respuesta le indicamos la opción *Two class unpaired*. La variable que indica el grupo ha de tomar valores 1 y 2. Vamos a tomar una tasa de falsamente rechazados o FDR muy pequeña. En concreto, $FDR = 0.001$.

```
data(golub,package="multtest")
fac0 = golub.cl +1
samfit = SAM(golub,fac0,resp.type="Two class unpaired",
    nperms=1000,fdr.output=.001)
```

¿Qué valor de Δ se ha utilizado?

```
samfit$del

##    delta
## 1.471802
```

¿Y qué genes encuentra el procedimiento significativo?

```
(sigtabla = samfit$siggenes.table)
```

Podemos comprobar en la salida anterior que diferencia entre genes *up* y genes *down*. ¿Qué es un gen up en este paquete? Indica asociación positiva entre expresión y la covariable y que nos indica el grupo (para golub recordad que la covariable es golub.cl que codifica ALL como 0 y AML como 1). En definitiva un gen up es que tiene una media mayor cuando la covariable es mayor, es decir, para los datos golub que la media en el grupo AML (valor de golub.cl 1) es mayor que la media en el grupo ALL (valor de golub.cl 0). Como es lógico los genes *down* son lo contrario. También podemos ver, para cada uno de ellos: el valor de d_i, su numerador y denominador, el cociente de las medias y el q-valor.

Tenemos también una ilustración gráfica en donde en abscisas nos muestra $\bar{d}_i$ y en ordenadas d_i. La línea en trazo continuo indica la línea donde $\bar{d}_i = d_i$. Las líneas en trazo discontinuo por arriba y por abajo indican los genes up y los genes down. En la figura 9.1 podemos ver los resultados.

```
plot(samfit)
```

```
## pdf
##   2
```

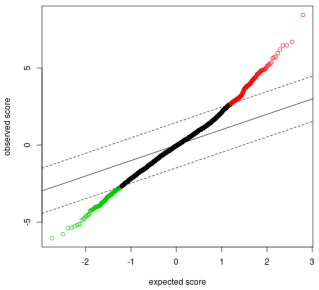


Figura 9.1: Método SAM utilizando [133, samr]. En negro los genes no significativos. En rojo los genes up que corresponden con asociación positiva con la covariable y en verde los que tienen asociación negativa.

Podemos ver también una exploración de posibles valores para Δ con

```
head(samfit$delta.table,n=1)

##         delta # med false pos 90th perc false pos
## [1,] 1.471802       0.5394952            3.776467
##      # called  median FDR 90th perc FDR     cutlo
## [1,]      336 0.001605641    0.01123948 -2.681051
##         cuthi
## [1,] 2.678885
```

9.1.4 SAM con otro tipo de covariables

Acabamos de utilizar este procedimiento en la comparación de dos grupos. Este mismo procedimiento se puede adaptar a los casos en que el vector y (covariable) nos indica la pertenencia de la muestra a más de un grupo o bien es un valor numérico.

El método es el mismo modificando las definiciones de d_i y s_i dadas en las ecuaciones 9.1 y 9.2.

Comparación de más de dos grupos Tenemos K grupos a comparar ($K > 2$) por lo que $y_j \in \{1, \ldots, K\}$. Supongamos que J_k denota los índices de las observaciones en grupo k.[4] Y supongamos que n_k es el cardinal de J_k ($\sum_{k=1}^{K} n_k = n$). Definimos d_i como

$$d_i = \frac{r_i}{s_i + s_0} \tag{9.3}$$

[4]Los conjuntos J_k son disjustos entre sí y su unión es el total de muestras, es decir, $\cup_{k=1}^{K} J_k = \{1, \ldots, n\}$.

con

$$r_i = \sqrt{\frac{\sum_{k=1}^{K} n_k}{\prod_{k=1}^{K} n_k} \sum_{k=1}^{K} n_k (\bar{x}_{i,J_k} - \bar{x}_{i\cdot})^2} \tag{9.4}$$

y

$$s_i = \sqrt{\frac{1}{\sum_{k=1}^{K}(n_k - 1)} \left(\sum_{k=1}^{K} \frac{1}{n_k} \right) \sum_{k=1}^{K} \sum_{j \in J_k} (x_{ij} - \bar{x}_{i,J_k})^2}. \tag{9.5}$$

Covariable continua Supongamos que los valores de y son valores numéricos en este caso definimos d_i como en 9.3 y los valores de r_i y s_i son

$$r_i = \frac{\sum_{j=1}^{n} y_j (x_{ij} - \bar{x}_{i\cdot})}{\sum_{j=1}^{n} (y_j - \bar{y})^2} \tag{9.6}$$

y

$$s_i = \frac{\hat{\sigma}_i}{\sqrt{\sum_{j=1}^{n} (y_j - \bar{y})^2}} \tag{9.7}$$

donde

$$\hat{\sigma}_i = \sqrt{\frac{\sum_{j=1}^{n} (x_{ij} - \hat{x}_{ij})^2}{n-2}}, \tag{9.8}$$

y

$$\hat{x}_{ij} = \hat{\beta}_{i0} + r_i y_j, \tag{9.9}$$

$$\hat{\beta}_{i0} = \bar{x}_{i\cdot} - r_i \bar{y}. \tag{9.10}$$

9.1.5 Ejercicios

Ejercicio 23

Utilizando los datos tamidata::gse20986 se pide:

1. Seleccionar las muestras correspondientes a iris y huvec.
2. Aplicar el método SAM y obtener el grupo de genes significativos.
3. Comparar con el grupo obtenido en el apartado 4 del ejercicio 21.

Ejercicio 24

Utilizando los datos tamidata::gse20986 se pide determinar los genes significativos cuando comparamos los cuatro grupos considerados en el estudio.

9.2 Limma

Hasta el momento nos hemos preocupado de estudiar la posible asociación entre el perfil de expresión de un gen y una sola covariable (o variable fenotípica). Nuestro mayor interés (aunque hemos considerado otros casos sobre todo con el método SAM) ha estado en el

caso en que la covariable nos define dos poblaciones a comparar dos tipos de cancer; una cepa mutante frente a una salvaje.

En este tema nos ocupamos de considerar la relación entre las expresiones del gen y más de una covariable. Situaciones habituales (y que consideramos aquí) será cuando tengamos las muestras clasificadas según dos criterios y queramos considerar el efecto simultáneo sobre la expresión. Ejemplos concretos de esta situación son:

1. Consideramos dos cepas (salvaje y mutante) y las observamos en varios instantes temporales. Supongamos que las muestras que observamos en cada instante temporal son independientes. Tenemos un diseño cruzado para cada gen.

2. Supongamos que tomamos muestras de un mismo individuo en dos instantes temporales (antes y después de un tratamiento por ejemplo). Además los individuos los tenemos clasificados en dos grupos. Para cada individuo tenemos una muestra apareada.

Los diseños experimentales anteriores son habituales en la literatura y los estudiamos en este tema. Y para ello se recurre a la teoría de modelos lineales adaptada a este contexto particular. El paquete básico en este tema es [126, limma]. [152]

[152] La viñeta de este paquete es, en sí misma, un buen libro y, sin duda, la referencia a consultar.

9.2.1 El modelo limma

El material de esta sección fue publicado en [125]. Lo que se hace es ajustar un modelo lineal para cada fila de la matriz de expresión considerando que tenemos valores independientes para las distintas muestras.[153] [154] Si trabajamos con datos de microarrays entonces supondremos que trabajamos con su logaritmo en base 2 ($\log_2$). Las expresiones aleatorias correspondientes al gen i-ésimo las denotamos por

$$\boldsymbol{X}_i = (X_{i1}, \ldots, X_{in})'.$$

Los correspondientes valores observados los denotaremos utilizando minúsculas, es decir;

$$\boldsymbol{x}_i = (x_{i1}, \ldots, x_{in})'.$$

[153] Y olvidando por tanto todo el proceso de corrección de fondo y normalización.

[154] En § 8 se incluye un repaso de la teoría de modelos lineales.

El vector X_i es el vector respuesta para el i-ésimo ajuste que vamos a realizar. [155] Denotamos, como es habitual, por $E(\boldsymbol{X}_i)$ el vector de medias del vector $\boldsymbol{X}_i$ que tiene en cada componente la media de la variable X_{ij}, es decir, $E\boldsymbol{X}_i = (EX_{i1}, \ldots, EX_{in})'$. Se asume:

[155] Asumimos que se ha preprocesado corrigiendo fondo y normalizando y que los datos están en escala $\log_2$.

1. $$E(\boldsymbol{X}_i) = A\boldsymbol{\alpha}_i \tag{9.11}$$
   siendo A una matriz de diseño de rango completo en columnas[5] y $\boldsymbol{\alpha}_i$ es un vector de coeficientes.

2. Suponemos que la matriz de covarianzas del vector aleatorio $\boldsymbol{X}_i$ que es proporcional (siendo desconocida la constante de proporcionalidad) a una matriz conocida, es decir, asumimos que
   $$var(\boldsymbol{X}_i) = \sigma_i^2 W_i, \tag{9.12}$$
   donde W_i es una matriz definida no negativa **conocida**. El vector $\boldsymbol{X}_i$ puede tener datos faltantes y la matriz W_i puede tener elementos en la diagonal principal nulos.

[5] Todas sus columnas linealmente independientes.

[156] § 8

Como es habitual en la teoría de modelos lineales[156] estamos interesados en ciertos contrastes sobre el vector de coeficientes $\boldsymbol{\alpha}_i$ que corresponden al diseño utilizado en la experiencia y, por lo tanto, evalúan problemas de interés desde un punto de vista biológico. Un contraste tiene la forma general

$$\beta_i = C'\boldsymbol{\alpha}_i.$$

Estamos interesados en contrastar si β_{ij} son nulas para los distintos j y para cada gen i, es decir, en la hipótesis nula: $H_0 : \beta_{ij} = 0$ vs $H_0 : \beta_{ij} \neq 0$.

Ajustamos un modelo lineal para cada gen y obtenemos los estimadores $\hat{\boldsymbol{\alpha}}_i$, el estimador s_i^2 de σ_i^2 y la matriz de covarianzas estimada

$$var(\hat{\boldsymbol{\alpha}}_i) = V_i s_i^2 \tag{9.13}$$

siendo V_i una matriz definida positiva que no depende de s_i^2. Los estimadores de los contrastes son

$$\hat{\boldsymbol{\beta}}_i = C'\hat{\alpha}_i,$$

y de su matriz de covarianzas es

$$var(\hat{\boldsymbol{\beta}}_i) = C'V_iCs_i^2.$$

No se asume necesariamente normalidad para $\boldsymbol{X}_i$ ni tampoco asumimos que el ajuste de los modelos lineales se hace mediante el procedimiento de los mínimos cuadrados. Pero **si** se asume que:

1. $\hat{\boldsymbol{\beta}}_i$ sí que tiene una distribución aproximadamente normal con vector de medias $\boldsymbol{\beta}_i$ y matriz de covarianzas $C'V_iC\sigma_i^2$.

2. También asumimos que s_i^2 siguen aproximadamente una distribución ji-cuadrado escalada.

La matriz V_i podría depender de $\boldsymbol{\alpha}_i$. Si esta dependencia se produce entonces consideramos V_i evaluada en $\hat{\boldsymbol{\alpha}}_i$ y que la dependencia puede ser ignorada al menos en una aproximación de primer orden.

Si denotamos por v_{ij} el j-ésimo elemento de la diagonal de $C'V_iC$ entonces todas las hipótesis indicadas previamente se traducen en que:

1. $\hat{\boldsymbol{\beta}}_{ij}|\boldsymbol{\beta}_{ij}, \sigma_i^2 \sim N(\boldsymbol{\beta}_{ij}, v_{ij}\sigma_i^2)$.

2. $s_i^2|\sigma_i^2 \sim \frac{\sigma_i^2}{d_i}\chi^2_{d_i}$ siendo d_i los grados de libertad residuales del modelo ajustado para el i-ésimo gen.

Asumiendo estas hipótesis se tiene que el siguiente estadístico

$$t_{ij} = \frac{\hat{\boldsymbol{\beta}}_{ij}}{s_i\sqrt{v_{ij}}}$$

aproximadamente tiene una distribución t de Student con d_i grados de libertad.

En lo que sigue se asumirá (y no tienen porqué ser cierto) que $\hat{\boldsymbol{\beta}}_i$ y s_i^2 son independientes para los distintos genes.

Tenemos muchos test simultáneos. Se trata de modelizar el comportamiento entre genes. Y para ello se opta por modelizar mediante distribuciones de probabilidad sobre los parámetros.[157] En concreto

[157] Tenemos un modelo bayesiano.

sobre σ_i^2 la siguiente distribución,

$$\frac{1}{\sigma_i^2} \sim \frac{1}{d_0 s_0^2}\chi^2_{d_0}.$$

Para un j dado, suponemos que β_{ij} es no nula con una probabilidad conocida

$$P(\beta_{ij} \neq 0) = p_j.$$

Notemos que p_j tiene el sentido de la proporción esperada de genes que se expresan diferencialmente. Para los coeficientes no nulos se asume

$$\beta_{ij}|\sigma_i^2, \beta_{ij} \neq 0 \sim N(0, v_{0j}\sigma_i^2).$$

Asumiendo el modelo jerárquico que acabamos de especificar se tiene que la media de la distribución a posteriori de $1/\sigma_i^2$ condicionada a s_i^2 viene dada por

$$E\left[\frac{1}{\sigma_i^2}|s_i^2\right] = \frac{1}{\tilde{s}_i^2},$$

con

$$\tilde{s}_i^2 = \frac{d_0 s_0^2 + d_i s_i^2}{d_0 + d_i}.$$

Podemos definir el estadístico t moderado[158] como

[158] Moderated t-statistic.

$$\tilde{t}_{ij} = \frac{\hat{\beta}_{ij}}{\tilde{s}_i^2\sqrt{v_{ij}}}.$$

Se demuestra que los t-estadísticos moderados $\tilde{t}_{ij}$ y las varianzas muestrales residuales s_i^2 se distribuyen independientemente. Y de hecho, tendremos que bajo la hipótesis nula $H_0 : \beta_{ij} = 0$, el t-estadístico moderado $\tilde{t}_{ij}$ sigue una distribución t de Student con $d_i + d_0$ grados de libertad.[159] Los grados de libertad que estamos añadiendo d_0 expresan la ganancia de información que obtenemos de utilizar todos los genes siempre **asumiendo el modelo jerárquico**. Los valores d_0 y s_0 que se suponen conocidos en lo previo serán estimados a partir de los datos.

[159] Ver sección 4 de [125].

Se consideran los odds a posteriori dados por

$$O_{ij} = \frac{P(\beta_{ij} \neq 0|\tilde{t}_{ij,s_i^2})}{P(\beta_{ij} = 0|\tilde{t}_{ij,s_i^2})}. \tag{9.14}$$

Estos odds prueban que tienen la siguiente expresión.

$$O_{ij} = \frac{p_j}{1-p_j}\left(\frac{v_{ij}}{v_{ij}+v_{0j}}\right)^{1/2}\left(\frac{\tilde{t}_{ij}^2 + d_0 + d_i}{\tilde{t}_{ij}^2 \frac{v_{ij}}{v_{ij}+v_{oj}} + d_o + d_i}\right)^{(1-d_0+d_i)/2} \tag{9.15}$$

Finalmente los autores proponen trabajar con los log-odds dados por

$$B_{ij} = \ln O_{ij}. \tag{9.16}$$

En el procedimiento que acabamos de comentar falta estimar los parámetros de las distribuciones a priori. Hemos de estimar d_0, s_0, v_{0j} (para todos los j) y las proporciones p_j. En principio, son valores que el propio usuario debiera de especificar. Representan la información previa y por lo tanto es la información que a priori y sin usar nada de la información muestral hemos de especificar. Es claro que esto no parece

muy factible. Los autores proponen lo que se conoce como un método empírico bayesiano y utilizan los propios datos para determinar estos valores.[160]

[160] Ver sección 6 de [125].

Utilizando los log-odds B podemos **ordenar** los genes de mayor a menor expresión diferencial. De hecho, si observamos la ecuación 9.15 y la definición 9.16 es fácil ver que, asumiendo que los d_i y los v_i no varían entre genes[161], entonces los log-odds B son funciones monótonas de $|\tilde{t}_{ij}$. En resumen, *l*a ordenación de los genes que obtenemos con los log-odds o con los estadísticos $|\tilde{t}_{ij}$ es la misma.

[161] O varían muy poco.

Es claro que los log-odds tienen una interpretación sencilla y, por ello, son útiles. Sin embargo, para decidir qué genes son significativos para el contraste en que estemos interesados es mejor opción trabajar con los t-estadísticos moderados.[162]

[162] Cuya definición es una especie de mezcla inteligente de Estadística clásica y bayesiana.

Trabajar con los t-estadísticos moderados, $\tilde{t}$, en lugar de con los log-odds tiene otra ventaja. Los $\tilde{t}$ dependen solamente de d_0 y s_0^2. Sin embargo, los log-odds dependen además de v_{0j} y de los p_j. Obviamente requerimos menos estimaciones previas de hiperparámetros.[163]

[163] Cosa nada desdeñable.

Lo más importante, sin duda,el t-estadístico moderado tiene la ventaja sobre el t-estadístico usual de que es menos probable que aparezcan valores muy grandes simplemente porque las varianzas muestrales infraestiman las varianzas poblaciones. Trabajamos con pequeñas muestras lo que hace que la estimación de la varianza sea poco precisa. Una infraestimación notable de esta varianza ocasiona que el estadístico sea muy grande aunque no tengamos una diferencia de medias grande. Este mismo problema se comenta en § 9.1. Allí se proponía añadir una constante (estimada a partir de los propios datos) a la desviación estándar de la diferencia de medias (el error estándar). ¿Esto es lo mismo? Es similar en espíritu pero no es lo mismo. Si los grados de libertad verifican $d_o < +\infty$ y $d_i > 0$ entonces los t-estadísticos moderados tienen la siguiente expresión

$$\tilde{t}_{ij} = \left(\frac{d_0 + d_i}{d_i}\right)^{1/2} \frac{\hat{\beta}_{ij}}{\sqrt{s^2_{*,i} v_{ij}}},$$

siendo

$$s^2_{*,i} = s_i^2 + \frac{d_0}{d_i} s_0^2.$$

Cada varianza muestral es incrementada un valor positivo en principio distinto para cada gen. La idea que vimos en § 9.1 consistía en considerar

$$t_{*,ij} = \frac{\hat{\beta}_{ij}}{(s_i + a)\sqrt{v_{ij}}}$$

donde a es un valor positivo determinado por un método ad-hoc. En este método lo que se incrementa es la desviación estándar y no la varianza. El t-estadístico moderado incrementa la varianza. Además el incremento puede ser distinto para cada gen.

¿Y si queremos contrastar más de un contraste al mismo tiempo? En lugar de t-estadísticos moderados tendremos F-estadísticos moderados. Se comprueba que los F-estadísticos moderados no son más que los F-estadísticos usuales en los cuales sustituimos las varianzas s_i^2 por $\tilde{s}_i^2$. Además su distribución es $F_i \sim F_{r,d_0+d_i}$, siendo r el número de contrastes linealmente independientes. Vemos que los grados de libertad del denominador también se incrementan.

En las siguientes secciones vamos a considerar distintos diseños experimentales y los analizaremos con [126, limma].[164]

[164] Este paquete es una de los fundamentales en el proyecto Bioconductor y de los más antiguos.

9.2.2 Datos gse44456 con limma

Leemos los datos tamidata::gse44456.

```
data(gse44456,package="tamidata")
```

Vamos a considerar la variable que nos da la tasa de crecimiento celular.

```
pData(gse44456)[,"postmortenint"]
```

```
##  [1] 16.75 27.00 29.00 24.00 24.00 24.00 17.00
##  [8] 12.00 21.00 36.00 19.00 36.00 68.00 46.00
## [15] 23.00 48.00 58.50 62.00 30.00 21.00 19.50
## [22] 24.00 59.50 50.00 22.00 37.00 41.00 16.00
## [29] 18.00 16.00  9.00 11.00 20.00 56.00 15.00
## [36] 43.00 11.00 32.00 21.50
```

Con objeto de entender lo que aporta el método limma respecto de un análisis utilizando modelos lineales en cada perfil de expresión para cada gen vamos a añadir algún código no necesario en un uso del paquete.

Cargamos Limma.

```
pacman::p_load(limma)
```

Determinamos la matriz de diseño.

```
design = model.matrix(~ pData(gse44456)[,"postmortenint"])
colnames(design) = c("intercept","postmortenint")
```

Podemos ver que la matriz `design`.

```
head(design)
```

```
##   intercept postmortenint
## 1         1         16.75
## 2         1         27.00
## 3         1         29.00
## 4         1         24.00
## 5         1         24.00
## 6         1         24.00
```

Ajustamos los modelos lineales.

```
fit = lmFit(gse44456,design)
```

```
fit1 = eBayes(fit)
```

Si la constante por la que multiplicamos growthrate es nula indicará que no hay una dependencia del nivel de expresión respecto de la tasa de crecimiento celular. Veamos los resultados.

```
topTable(fit1,coef=2,adjust ="BH")
```

```
##                logFC   AveExpr         t
## 7898750 -0.011508519  6.747570 -6.443278
## 8170468  0.013115884  6.012464  5.689956
## 7969651 -0.010989325  7.051056 -5.687741
## 8111415  0.008178560  2.540914  5.065649
## 7895171  0.012638970  8.895633  4.980637
## 7959012  0.012611550  4.428257  4.935664
## 7972269 -0.009755820  4.360153 -4.919346
## 8152664 -0.015744665  4.187651 -4.870722
## 7896252 -0.008523726 11.280638 -4.834066
## 7921533 -0.013989202  8.399406 -4.826789
##              P.Value   adj.P.Val        B
## 7898750 1.016754e-07 0.003385485 6.398649
## 8170468 1.196673e-06 0.013378383 3.937214
## 7969651 1.205368e-06 0.013378383 3.930008
## 8111415 9.095965e-06 0.060272779 1.924620
## 7895171 1.195878e-05 0.060272779 1.654193
## 7959012 1.381685e-05 0.060272779 1.511583
## 7972269 1.455932e-05 0.060272779 1.459918
## 8152664 1.701344e-05 0.060272779 1.306232
## 7896252 1.912948e-05 0.060272779 1.190642
## 7921533 1.957948e-05 0.060272779 1.167723
```

Aunque alguno de los p-valores originales pudieran (marginalmente) considerarse como significativos cuando corregimos por comparaciones múltiples por el método de Benjamini-Hochberg no lo son.

9.2.3 Un diseño cruzado con limma

Seguimos utilizando tamidata::gse44456.

```
pacman::p_load("limma")
data(gse44456,package="tamidata")
```

Observamos los datos fenotípicos o metadatos.

```
head(pData(gse44456))
```

```
##                                case gender age
## GSM1085665_HE32H001.CEL      control   male  68
## GSM1085666_HE32H003.CEL    alcoholic   male  51
## GSM1085667_HE32H004.CEL      control   male  50
## GSM1085668_HE32H005.CEL    alcoholic   male  50
## GSM1085669_HE32H006_2_.CEL   control   male  56
## GSM1085670_HE32H007_2_.CEL alcoholic   male  59
##                            cirrhosis smoker
## GSM1085665_HE32H001.CEL           No     No
## GSM1085666_HE32H003.CEL           No    Yes
## GSM1085667_HE32H004.CEL           No     No
## GSM1085668_HE32H005.CEL          Yes    Yes
## GSM1085669_HE32H006_2_.CEL        No    Yes
## GSM1085670_HE32H007_2_.CEL        No     No
```

```
##                              postmortenint   pH
## GSM1085665_HE32H001.CEL             16.75 6.59
## GSM1085666_HE32H003.CEL             27.00 5.58
## GSM1085667_HE32H004.CEL             29.00 6.68
## GSM1085668_HE32H005.CEL             24.00 6.59
## GSM1085669_HE32H006_2_.CEL          24.00 6.53
## GSM1085670_HE32H007_2_.CEL          24.00 6.57
##                                 batch
## GSM1085665_HE32H001.CEL    Batch 1
## GSM1085666_HE32H003.CEL    Batch 1
## GSM1085667_HE32H004.CEL    Batch 2
## GSM1085668_HE32H005.CEL    Batch 2
## GSM1085669_HE32H006_2_.CEL Batch 1
## GSM1085670_HE32H007_2_.CEL Batch 1
```

Analizamos la posible dependencia con el alcoholismo recogida en la covariable `case`

```
alcoholism = pData(gse44456)[,"case"] # Simplificamos el código
design = model.matrix(~ 0 + alcoholism)
colnames(design) = c("control","alcoholic")
```

Ajustamos el modelo.

```
fit = lmFit(gse44456,design)
```

```
(contrast.matrix = makeContrasts(dif = control - alcoholic,levels = design))
```

```
##            Contrasts
## Levels      dif
##   control     1
##   alcoholic  -1
```

Estimamos.

```
fit2 = contrasts.fit(fit,contrast.matrix)
fit2 = eBayes(fit2)
```

Veamos cuáles son significativos.

```
topTable(fit2,coef=1,adjust="BH")
```

```
##              logFC  AveExpr         t
## 7927186 -0.5424921 7.826424 -5.914411
## 8125919 -1.1358866 8.313619 -5.628792
## 8021081 -1.2885800 8.593822 -5.481851
## 7961595  0.5322101 4.180459  5.440839
## 7995838 -0.9825655 8.540374 -5.096932
## 8130867  0.5858452 7.566787  5.083113
## 8114814  0.3685955 7.694306  5.003841
## 7922889  0.5172199 8.438433  4.946850
## 8021741  0.7651623 9.183204  4.720399
```

```
## 8099476  0.5231415 5.584413  4.658028
##              P.Value  adj.P.Val        B
## 7927186 5.718213e-07 0.01903993 5.029195
## 8125919 1.455678e-06 0.02236997 4.307423
## 8021081 2.351231e-06 0.02236997 3.935029
## 7961595 2.687326e-06 0.02236997 3.831026
## 7995838 8.199033e-06 0.04757580 2.959048
## 8130867 8.572989e-06 0.04757580 2.924055
## 8114814 1.106824e-05 0.05264847 2.723446
## 7922889 1.329382e-05 0.05533056 2.579390
## 8021741 2.741801e-05 0.10143750 2.008880
## 8099476 3.342454e-05 0.10201553 1.852423
```

En el segunda análisis nos fijamos en dos covariables, el alcoholismo y el sexo de la persona. Es un diseño factorial 2×2. Veamos cuántos datos tenemos en cada celda.

```
table(pData(gse44456)[,"case"],pData(gse44456)[,"gender"])
```

```
##
##             male female
##   control     13      6
##   alcoholic   14      6
```

En un diseño como este las preguntas habituales son las siguientes:

1. ¿Qué genes muestran un comportamiento diferenciado o relacionado con el alcoholismo?
2. ¿Qué genes se comportan de un modo distinto para hombres y mujeres?
3. ¿Para qué genes los cambios en su expresión según el alcoholismo son distintos en cada uno de los sexos?

En jerga estadística hablaríamos de los efectos principales de los factores y de la posible interacción.

Una aproximación simple y efectiva es construir un solo factor con todas las combinaciones de los factores.

```
casegender = vector("list",ncol(gse44456))
for(i in seq_along(casegender))
  casegender[[i]] = paste(pData(gse44456)[,"case"][i],
                          pData(gse44456)[,"gender"][i],sep="")
casegender = factor(unlist(casegender))
```

Consideremos la siguiente matriz de diseño.

```
design = model.matrix(~  0 + casegender)
colnames(design) = levels(casegender)
```

Podemos ver que cada coeficiente corresponde con la expresión media para la correspondiente combinación de factores.

```
fit = lmFit(gse44456,design)
```

Construimos los contrastes en que estamos interesados.[165]

[165] Un detalle: los nombres de los niveles no pueden empezar por un número.

```
cont.matrix = makeContrasts(
    dif1 = controlmale - alcoholicmale,
    dif2 = controlfemale - alcoholicfemale,
    dif12 = (controlmale - alcoholicmale)- (controlfemale - alcoholicfemale),
    levels = design)
fit2 = contrasts.fit(fit,cont.matrix)
fit2 = eBayes(fit2)
```

Podemos ejecutar el siguiente código y podemos comprobar que ningún contraste es significativo.

```
topTable(fit2,coef=1,adjust="BH")
topTable(fit2,coef=2,adjust="BH")
topTable(fit2,coef=3,adjust="BH")
```

9.3 Un comentario técnico importante

En este tema hemos trabajado con p-valores que están relacionados entre sí. Si denotamos por P_i el valor aleatorio de el p-valor correspondiente al contraste de H_i frente a K_i entonces su distribución (si asumimos distribución continua para el estadístico como en el test de la t de Student), bajo la hipótesis nula H_i, será uniforme en el intervalo unitario $[0,1]$, es decir, $P_i \sim U(0,1)$. Hay procedimientos de corrección de estos p-valores que consideran que los P_i son independientes. A priori sabemos que esto no es cierto. Los genes tienen interdependencias que se traducen en dependencias entre las distribuciones de los P_i. Sin embargo, hay otro efecto también muy destacable y no tan evidente a priori. Aquellos genes que tienen una expresión mayor son detectados como diferencialmente expresados con más probabilidad. La potencia del test es mayor.

Capítulo 10

Expresión diferencial con datos RNASeq

10.1 Introducción

Tenemos datos de expresión de gen obtenidos mediante la técnica conocida como RNA-seq. Pero, ¿qué tenemos realmente? ¿Con qué vamos a trabajar?

Ya hemos realizado todo el preprocesado. Ya hemos alineado y contado lecturas asociadas **a gen o a transcritos**. Nuestra información muestral es una matriz de expresión **observada** $\boldsymbol{x} = [x_{ij}]_{i=1,\ldots,N;j=1,\ldots,n}$ donde hemos considerado N genes y tenemos n muestras. Además tendremos covariables, variables fenotípicas o metadatos.[166] Estas covariables son otra matriz $n \times p$ donde n es el número de muestras y p es el número de covariables de las que disponibles sobre las muestras. Tendremos $\boldsymbol{y} = [y_{jk}]_{j=1,\ldots,n;k=1,\ldots,p}$. Habitualmente $p = 1$ y $\boldsymbol{y} = (y_1, \ldots, y_n)'$ es simplemente un vector.

Como indicamos el estudio se puede realizar a nivel de transcrito o a nivel de gen. Si utilizamos sus abreviaturas en inglés podemos hablar de DTE (differential transcript expression) y DGE (differential gene expression). En lo que sigue abreviamos de este modo.

[166] Un estadístico dice covariable, un bioinformático dice variable fenotípica y un moderno de los de los Big Data diría cualquier cosa, por qué no, metadatos.

10.2 Una muestra por condición

¿Y si no tenemos réplicas en cada condición que queremos comparar? Queremos comparar dos condiciones y disponemos de una muestra en cada una de las condiciones. ¿Qué podemos hacer para comparar los conteos para cada gen?[167] En esta (mala) situación la información que tenemos para cada gen la podemos recoger en una tabla de contingencia 2×2 (10.1). En esta tabla y_j recoge el número de lecturas alineadas sobre el gen que estamos estudiando en la muestra j (con $j = 1, 2$). Los tamaños de las librerías en cada muestra son m_j (con $j = 1, 2$).

Vamos a considerar dado el total de conteos asociados al gen i-ésimo, esto es, consideramos constante el conteo $x_{i1} + x_{i2}$.[168] El gen que consideramos tiene un conteo aleatorio X_{i1} en la primera muestra (por ejemplo, correspondiente a caso). Su distribución de probabilidad será binomial con $x_{i1} + x_{i2}$ pruebas y una probabilidad de éxito (lectura procedente de la primera muestra) desconocida p. Si no hay

[167] Lo primero sería pensar en tomar alguna muestra más por condición que los milagros solamente son los jueves.http://es.wikipedia.org/wiki/Los_jueves,_milagro y https://www.youtube.com/watch?v=-yDNya7XP-4.

[168] Que obviamente es aleatorio. Trabajamos condicionados a ese valor.

Tabla 10.1: Una muestra por condición en donde la primera sería caso y la segunda control.

	Muestra 1 (Caso)	Muestra 2 (Control)	Total
Gen i	x_{i1}	x_{i2}	$x_{i\cdot} = x_{i1} + x_{i2}$
Resto	$m_1 - x_{i1}$	$m_2 - x_{i2}$	$m_1 + m_2 - x_{i\cdot}$
Total	m_1	m_2	$m_1 + m_2$

diferencias entre las muestras el valor de p debiera de coincidir con $m_1/(m_1+m_2)$, es decir, debe coincidir la proporción con la proporción que la primera muestra representa en el total de las dos muestras. Por tanto, evaluar la expresión diferencial del gen se traduce en un contraste sobre p la probabilidad de que la lectura corresponda al caso. Una primera referencia en este sentido es [73].

En edgeR::binomTest utilizan como p-valor la suma de las probabilidades de los conteos que tienen una probabilidad menor o igual al observado bajo la hipótesis nula.

Consideremos los datos tamidata::PRJNA297664. Nos quedaremos con una muestra "wild" y otra "SEC66 deletion". Compararemos entre sí estas dos muestras.

```
pacman::p_load("SummarizedExperiment")
data(PRJNA297664,package="tamidata")
se = PRJNA297664
rm(PRJNA297664)
```

La variable que indica el tipo es `treatment`.

```
colData(se)[,"treatment"]

## [1] Wild           Wild           SEC66 deletion
## [4] Wild           SEC66 deletion SEC66 deletion
## Levels: Wild SEC66 deletion
```

Vamos a comparar las muestras 1 y 3 con la función edgeR::binomTest.

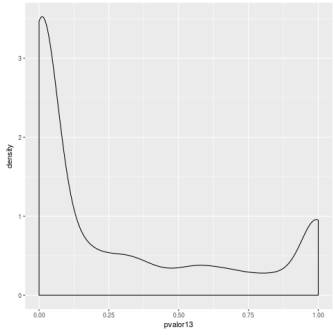


Figura 10.1: Densidad de p-valores.

```
pacman::p_load("edgeR")
pvalor13 = edgeR::binomTest(assay(se)[,1], assay(se)[,3])
```

Podemos hacernos una idea de cómo son estos p-valores con una estimación de su función de densidad (figura 10.1).

También podemos aplicar para el mismo problema el test de Fisher. En cualquier caso, **solamente tiene sentido hacer este análisis cuando tenemos una muestra en cada condición**.[169]

¿Qué hacemos con más de una muestra? ¿Realizar todas las comparaciones dos a dos? No. Obviamente hay que buscar procedimientos más adecuados.

¿Se puede realizar algo similar cuando tenemos varias muestras por cada una de las dos condiciones? Es tentador simplemente sumar los conteos de cada gen en cada condición. Tendríamos el total de lecturas por gen y condición. Sin embargo, si consideramos la tabla 10.1 correspondiente estaríamos ignorando la variabilidad intra condición de los conteos correspondientes. En este caso hemos de acudir a alguno de los procedimientos que vemos en la siguientes secciones.

[169] Y tampoco mucho.

De porqué una muestra por condición no es buena consejera. Vamos a evaluar de un modo empírico porqué necesitamos más muestras por condición. Porqué esa práctica (más frecuente de lo que se debiera) de tener una muestra por condición no se debe de utilizar. Siguiendo con PRJNA297664 tenemos 9 posibles pares formados por una muestra por condición. Vamos a calcular los p-valores para cada una de estas nueve posibles comparaciones y ver cómo varían estos p-valores.

```
pares = rbind(c(1,3),c(1,5),c(1,6),c(2,3),c(2,5),c(2,6),c(4,3),
    c(4,5),c(4,6))
aux = function(rr) binomTest(assay(se)[,rr[1]], assay(se)[,rr[2]])
pvalores = apply(pares,1,aux)
```

Por cada gen tenemos el p-valor resultado de comparar cada uno de los pares. Tenemos pues 9 p-valores por gen. Calculamos el p-valor medio, el mínimo y el máximo y los ordenamos de menor a mayor. Finalmente representamos en abscisas el p-valor medio y en ordenadas dos líneas correspondientes al mínimo y al máximo. Dada la gran cantidad de puntos hemos de representar una versión suavizada del mínimo y máximo.

```
png("figures/RNASeqDE10.png")
pvalores.mean = apply(pvalores,1,mean) ## Media de p-valores
pvalores.min = apply(pvalores,1,min)  ## Mínimo de p-valores
pvalores.max = apply(pvalores,1,max)  ## Máximo de p-valores
df = data.frame(pvalores.mean,pvalores.min,pvalores.max)
df=df[sort(pvalores.mean,index.return=TRUE)$ix,] ## Ordenamos
df1 = reshape2::melt(df,id="pvalores.mean")
pp = ggplot(df1,aes(x=pvalores.mean,y=value,color=variable)
pp + geom_smooth()  # Suavizamos  mínimo y máximo
dev.off()
```

En la figura 10.2 tenemos el resultado. Podemos ver la tremenda variabilidad que tenemos.

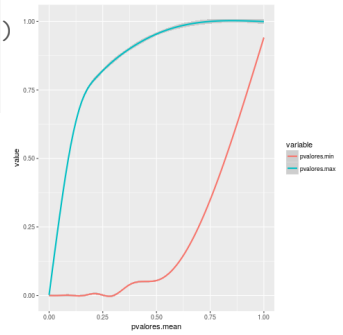


Figura 10.2: p-valores.

10.3 edgeR

Este paquete implementa los procedimientos propuestos en [117, 116][170]

10.3.1 Estimación de una dispersión común

Consideramos una característica en n muestras (o librerías) de tamaños distintos.[171] Sea m_i el tamaño de la i-ésima librería (total de lecturas). Sea λ la proporción que hay en una librería cualquiera de la característica en que estamos interesados. *Vamos a asumir* que Y_i tiene una distribución binomial negativa con media $\mu_i = m_i\lambda$ y con dispersión ϕ. Y tiene una distribución binomial negativa con media μ y dispersión ϕ, $Y \sim BN(\mu, \phi)$, si su función de probabilidad es $P(Y = y|\mu,\phi) = \frac{\Gamma(y+\phi^{-1})}{\Gamma(\phi^{-1})\Gamma(y+1)}\left(\frac{1}{1+\mu\phi}\right)^{\phi^{-1}}\left(\frac{\mu}{\phi^{-1}+\mu}\right)^{y}$ para $y = 0, 1, \ldots$. Su media es $E(Y) = \mu$ y su varianza $var(Y) = \mu + \phi\mu^2$. Ver § 3.8.2.

[170] Hay que leer primero [117] y luego [116] pues es la secuencia temporal de los trabajos aunque se publicaron en orden invertido.

[171] Son réplicas de una misma experimentación con tamaños distintos.

Supongamos el caso ideal (e irreal) en que todas las librerías tienen el mismo tamaño: $m_i = m$ para $i = 1, \ldots, n$. Bajo esta condición tenemos que $Z = \sum_{i=1}^{n} Y_i \sim NB(nm\lambda, \phi/n)$. Además la logverosimilitud condicionada al valor de $Z = z$ no depende de λ y podemos estimar el valor de ϕ. Por tanto, se estimaría ϕ maximizando la verosimilitud condicionada dada por

$$l_{\mathbf{y}|z} = \left[\sum_{i=1}^{n} \log\Gamma(y_i + 1/\phi)\right] + \log\Gamma(n/\phi) - \log\Gamma(z + n/\phi) - n\log\Gamma(1/\phi), \quad (10.1)$$

donde $\mathbf{y} = (y_1, \ldots, y_n)'$ son los conteos observados. Si los tamaños de las librerías son distintos la verosimilitud no tiene una expresión simple. La idea en [117] es modificar los valores observados y_i y generar unos nuevos datos en que los tamaños de las librerías coincidan, los *pseudodatos*. El tamaño común será la media geométrica de los tamaños observados, $m^* = \sqrt[n]{\prod_{i=1}^{n} m_i}$. ¿Cómo transformamos los conteos y_i? El procedimiento propuesto es:

1. Inicializamos ϕ (por ejemplo, con el estimador maxímo verosímil condicionado sin realizar ningún ajuste).

2. Dado el valor estimado de ϕ, estimamos λ maximizando la verosimilitud para el valor estimado de la dispersión.

3. Suponemos que cada conteo y_i es un valor observado de una distribución binomial negativa con media $m_i\lambda$ y parámetro de dispersión ϕ. Calculamos los percentiles

$$p_i = P(Y < y_i | m_i\lambda, \phi) + \frac{1}{2}P(Y = y_i | m_i\lambda, \phi), \quad (10.2)$$

   para $i = 1, \ldots, n$.

4. Suponemos ahora una distribución binomial negativa con media $m^*\lambda$ y dispersion ϕ. Determinamos qué valor sería el percentil de orden p_i en la nueva distribución. Ya no tiene porqué ser un dato entero e incluso puede ser negativo. Los pseudodatos para un mismo gen tienen aproximadamente la misma distribución.

5. Estimamos la dispersión ϕ con los pseudodatos utilizando la veromilitud condicionada a la (pseudo) suma total de un gen.

6. Repetimos desde 2 hasta 5 hasta que converja ϕ.

A este procedimiento de estimación de ϕ y λ los autores lo denominan *máxima verosimilitud condicionada ajustada por cuantiles*.[172] Lo fundamental es entender que estamos modificando los datos para conseguir un tamaño común de las librerías, una misma profundidad de secuenciación.

[172] Quantile-adjusted conditional maximum likelihood (qCML).

10.3.2 Un test exacto

Tenemos dos condiciones a comparar.

Estimación de la dispersión común. Vamos a asumir en un primer momento que el parámetro de dispersión es común. Tenemos y_{ijk} el conteo para la muestra k en la condición j del gen i donde $j = 1, 2$ y $k = 1, \ldots, n_j$. El tamaño de la librería de la condición j en el gen i es m_{ij} Supongamos que aplicamos el método qCML dentro de cada condición j de modo que por condición tenemos un tamaño de librería común. Con los pseudodatos (que denotamos también y_{ijk} podemos condicionar a la suma total por gen dentro de cada clase. **Como el parámetro de dispersión se asume el mismo para todos los genes y condiciones** podemos considerar la logverosimilitud condicionada a $z_{ij} = y_{ij\cdot} = \sum_{k=1}^{n_j} y_{ijk}$ para la dispersión ϕ dada por

$$l_i(\phi) = \sum_{j=1}^{2} \left(\sum_{k=1}^{n_j} \log\Gamma(y_{ijk} + \phi^{-1}) + \log\Gamma(n_j\phi^{-1}) - \log\Gamma(z_{ij} + n_j\phi^{-1}) - n_j \log\Gamma(\phi^{-1}) \right). \quad (10.3)$$

En consecuencia la logverosimilitud común es[173]

$$l(\phi) = \sum_{i=1}^{N} l_i(\phi). \quad (10.4)$$

[173] Sería más correcto hablar de pseudoverosimilitud ya que los distintos genes son dependientes.

La estimación de ϕ en § 10.3.1 se obtiene maximizando la función dada en la ecuación 10.4. Denotamos a este estimador común por $\hat{\phi}_C$ y es el usado en lo que sigue.

Contraste de hipótesis. Tenemos dos condiciones a comparar y estamos asumiendo

$$EY_{ijk} = m_{ij}\lambda_{ij},$$

para cualquier i, j, k siendo Y_{ijk} el conteo aleatorio del gen i en la condición j y en la muestra k. Si consideramos la hipótesis nula de que no hay diferencia entre los valores de λ, es decir, el contraste siguiente:

$$\begin{aligned} H_i: &\quad \lambda_{i1} = \lambda_{i2}, \\ K_i: &\quad \lambda_{i1} \neq \lambda_{i2}. \end{aligned}$$

Bajo la hipótesis nula el valor λ no depende de la condición. Aplicamos el método qCML a **todas** las muestras (tenemos $n = n_1 + n_2$). Como es habitual denotamos los nuevos pseudodatos como los originales. El tamaño de las librerías es la media geométrica de los tamaños originales y la denotamos por m^*.[174] Estos nuevos psedodatos tienen una distribución común. En concreto para el gen i denotamos por y_{ijk} el conteo para la muestra k en la condición j donde $j = 1, 2$ y $k = 1, \ldots, n_j$. El tamaño común de todas las librerías es m^*. Si consideramos la variable aleatoria Y_{ijk} (de la cual y_{ijk} sería el valor observado) entonces

[174] Importante, los pseudodatos no son los mismos utilizados para estimar la dispersión común.

$$EY_{ijk} = m^*\lambda_{ij},$$

para cualquier i, j, k. Bajo la hipótesis nula H_0 no tendríamos diferencia en el valor de λ entre las condiciones y sería un valor común $\lambda_i = \lambda_{i1} = \lambda_{i2}$ para el gen i. Utilizando el estimador previamente

calculado $\hat{\phi}_C$ y los pseudatos (que obtienen un tamaño de librería común) y_{ijk} con $k = 1, \ldots, n_j$ podemos estimar λ_i. Simplemente el estimador máximo verosímil no es más que el cociente de la suma de los conteos dividido por la suma de los tamaños de las librerías. Tenemos las estimaciones $\hat{\lambda}_i$ y $\hat{\phi}_C$.

Bajo la hipótesis nula de no diferencia entre grupos tendríamos que $Y_{ij\cdot} = \sum_{k=1}^{n_j} Y_{ijk} \sim NB(n_j m^* \hat{\lambda}_i, \hat{\phi}_C/n_j)$ para $j = 1, 2$. Además $Y_{i1\cdot}$ e $Y_{i2\cdot}$ son independientes. La suma $Y_{i1\cdot} + Y_{i2\cdot}$ también tiene una distribución binomial negativa: $Y_{i1\cdot} + Y_{i2\cdot} = \sum_{i=1}^{2} \sum_{k=1}^{n_j} Y_{ijk} \sim NB((n_1 + n_2)m^* \hat{\lambda}_i, \hat{\phi}_C/(n_1 + n_2))$. Podemos considerar la distribución condicionada del vector aleatorio $(Y_{i1\cdot}, Y_{i2\cdot})$ a la suma $Y_{i1\cdot} + Y_{i2\cdot}$ y considerar las probabilidades de los conteos conjuntos que *son menos probables que el observado*. La suma de estas probabilidades nos daría el p-valor del test.[175]

[175] Es un test similar al test de Fisher bilateral en donde las probabilidades hipergeométricas son sustituidas por probabilidades por la distribución binomial negativa. Ver [3, Página 93].

10.3.3 Dispersiones posiblemente distintas

En el test exacto que hemos visto en § 10.3.2 hemos asumido una dispersión común ϕ y se ha estimado su valor utilizando los conteos asociados a todos los genes y condiciones bajo las cuales se han observado a estos genes. Bajo la hipótesis nula de un misma media, y asumiendo la dispersión común esto tenía sentido. Es, sin duda, una hipótesis bastante exigente. Una misma dispersión sobre una cantidad grande de genes no es muy razonable.[176] En [116] proponen no asumir un mismo valor para cada gen y considerar que las dispersiones dependen del gen. Tenemos ahora, en lugar del valor común ϕ, un valor (posiblemente) distinto para el gen i, ϕ_i. ¿Y cómo lo estimamos? Proponen un compromiso entre la contribución que hacen los conteos del i-ésimo gen, l_i (ecuación 10.3), a la logverosimilitud global, l (ecuación 10.4), y la propia logverosimilitud global. En concreto hablan de la logverosimilitud condicionanda ponderada definida como

[176] Buena desde el punto de vista del modelo y su estimación pero nos aleja de los datos.

$$WL(\phi_i) = l_i(\phi_i) + \alpha l(\phi_i), \tag{10.5}$$

siendo α el peso que se da a la verosimilitud global. Es una función que propone un compromiso entre considerar l como función a maximizar considerando la misma contribución a todos los genes y l_i en donde solamente consideramos los conteos del propio gen. El problema fundamental a considerar ahora es la elección del valor de α. Un mayor α nos aproxima a un valor común para la dispersión y un menor valor nos dará estimaciones distintas de la dispersión para cada gen. ¿En qué punto nos quedamos? En [116, Sección, 3.2, pág. 2883] dan una motivación[177]

[177] No un criterio de optimalidad. Simplemente argumentan lo razonable de la elección.

Para explicar cómo lo hacen hemos de considerar la **función score** definida como

$$S_i(\phi) = \frac{\partial l_i(\phi)}{\partial \phi} \tag{10.6}$$

y la información esperada definida como

$$I_i(\phi) = E(J_i), \text{ siendo } J_i = -\frac{\partial^2 l_i(\phi)}{\partial \phi^2}. \tag{10.7}$$

El procedimiento de estimación de α y la posterior estimación de ϕ_i es el siguiente:

1. Estimamos la dispersión común maximizando la verosimilitud global l: $\hat{\phi}_0$.
2. Evaluamos para cada gen i: $S_i(\hat{\phi}_0)$ y $I_i(\hat{\phi}_0)$.
3. Estimamos τ_0 resolviendo la ecuación

$$\sum_{i=1}^{N}\left[\frac{S_i(\hat{\phi}_0)}{I_i(\hat{\phi}_0)(1+I_i(\hat{\phi}_0)\tau_0^2)}-1\right]=0.$$

   Si $\sum_{i=1}^{N} S_i^2(\hat{\phi}_0)/I_i(\hat{\phi}_0) < N$ entonces $\tau_0 = 0$.
4. Fijamos

$$1/\alpha = \tau_0^2 \sum_{i=1}^{N} I_i(\hat{\phi}_0).$$

5. Una vez estimado α en el paso anterior, maximizamos la función $WL(\phi_i)$ definida en la ecuación 10.5.

Finalmente los autores utilizan en lugar de la información esperada $I_i(\hat{\phi}_0)$ una aproximación que consiste en utilizar la observada (entendiendo por observada la correspondiente a los conteos dados). Es decir, sustituyen $I_i(\hat{\phi}_0)$ por $J_i(\hat{\phi}_0)$. Además comentan que trabajan con $\delta = \phi/(\phi+1)$.[178]

[178] Lo cual es irrelevante. Estimamos δ estimamos ϕ pues hay una correspondencia 1-1.

10.3.4 edgeR: PRJNA297664

En esta sección analizamos la posible expresión diferencial de los datos tamidata::PRJNA297664.

```
data(PRJNA297664,package="tamidata")
```

```
pacman::p_load("edgeR","SummarizedExperiment")
dge = DGEList(counts=assay(PRJNA297664),
    group=colData(PRJNA297664)[,"treatment"])
dge.c = estimateCommonDisp(dge) ##Estimamos dispersión común
dge.c$common.dispersion

## [1] 0.01170892

dge.t =  estimateTagwiseDisp(dge.c) ##Dispersiones por gen
et.c = exactTest(dge.c)
et.t = exactTest(dge.t)
```

Determinamos los genes significativos.

```
topTags(et.c)

## Comparison of groups:  SEC66 deletion-Wild
##                logFC   logCPM        PValue
## YBR171W   -10.150220 6.114922 2.128721e-258
## YCR021C    -1.928779 8.463733  4.873506e-47
## YBR054W    -1.878579 7.156683  3.518798e-43
## YGL255W    -1.846348 7.518466  1.930761e-42
```

```
## YNR034W-A  -2.176676 4.670571  1.175800e-40
## YBR093C    -1.694281 8.563323  2.731791e-37
## YFR053C     1.621598 6.384697  2.625927e-31
## YER150W    -1.560027 5.479642  2.325258e-26
## YDR171W    -1.383170 7.792428  2.142035e-25
## YDR214W     1.362108 7.982939  1.024220e-24
##                     FDR
## YBR171W   1.516926e-254
## YCR021C    1.736430e-43
## YBR054W    8.358319e-40
## YGL255W    3.439650e-39
## YNR034W-A  1.675750e-37
## YBR093C    3.244457e-34
## YFR053C    2.673194e-28
## YER150W    2.071224e-23
## YDR171W    1.696016e-22
## YDR214W    7.298590e-22
```

```
topTags(et.t)
```

```
## Comparison of groups:  SEC66 deletion-Wild
##                logFC    logCPM        PValue
## YBR171W   -10.150503  6.114922 5.636512e-296
## YGL255W    -1.846051  7.518466  5.459134e-30
## YBR093C    -1.694086  8.563323  5.843701e-27
## YNR034W-A  -2.174452  4.670571  7.411466e-22
## YDR214W     1.362104  7.982939  2.334672e-20
## YLR109W     1.240894  9.510958  4.583075e-20
## YHR215W    -1.208860  9.533270  2.172522e-18
## YAR071W    -1.211709  9.219219  3.764430e-18
## YMR186W     1.109256 11.622933  9.483967e-18
## YKL161C     1.100257  5.723199  1.070358e-16
##                     FDR
## YBR171W   4.016579e-292
## YGL255W    1.945090e-26
## YBR093C    1.388074e-23
## YNR034W-A  1.320353e-18
## YDR214W    3.327375e-17
## YLR109W    5.443166e-17
## YHR215W    2.211627e-15
## YAR071W    3.353166e-15
## YMR186W    7.509195e-15
## YKL161C    7.627369e-14
```

10.4 Un método de cuasi-verosimilitud

En esta sección tratamos el método propuesto en [84] e implementado en [85, QuasiSeq].

Utilizamos en lo que sigue la notación de [84].

10.5 SAMseq

Este método fue propuesto en [79] y está implementado en el paquete [133, samr]. Supongamos el caso con dos grupos. Denotamos como siempre por x_{ij} el conteo de la i-ésima característica en la j-ésima muestra ($i = 1, \ldots, N$ y $j = 1, \ldots, n$). El tamaño de la librería o profundidad de secuenciación será $m_j = \sum_{j=1}^{N} x_{ij}$. Como siempre el valor observado es x_{ij} mientras que el valor aleatorio (antes de observar los datos) lo denotamos con X_{ij}. Los índices correspondientes a los dos grupos de muestras los denotamos por C_1 y C_2. Estos conjuntos de índices constituyen una partición de $\{1, \ldots, n\}$. Suponemos que las clases C_1 y C_2 la componen n_1 y n_2 elementos (con $n_1 + n_2 = n$).

Supongamos, en un primer momento, que todas las profundidades de secuenciación son iguales $m_1 = \ldots = m_n = m$. Fijamos la característica i. Ordenamos (de menor a mayor) los conteos $x_{i1}, \ldots, x_{in}$. La posición de x_{ij} en $\boldsymbol{x}_{i*} = (x_{i1}, \ldots, x_{in})$ la denotamos por $r_{ij}(\boldsymbol{x}_{i*})$. Si consideramos el valor aleatorio X_{i*} entonces tenemos la variable aleatoria $R_{ij}(\boldsymbol{X}_{i*})$. El estadístico de Mann-Whitney-Wilcoxon[179] sería

$$T_i = \sum_{j \in C_1} R_{ij}(\boldsymbol{X}_{i*}) - \frac{n_1(n+1)}{2}. \tag{10.8}$$

[179] A veces se le llama de Mann-Whitney, otras de Wilcoxon y otras con los tres nombres.

Se supone que no tenemos empates en $\boldsymbol{X}_{i*}$. Si T_i es muy grande significa que en la característica i los valores en la primera clase (C_1) tienden a ser mayores que en la segunda (C_2). Un valor muy pequeño indica lo contrario. Notemos que T_i puede tomar valores negativos. En general, si el valor absoluto $|T_i|$ es muy grande, lo que tenemos es que ambos grupos de valores son claramente distintos, uno mayor que el otro. Valores próximos a cero indican que no hay diferencias entre los grupos. Hasta aquí no hay problema ninguno. Simplemente se propone utilizar un test de Mann-Whitney-Wilcoxon para comparar los órdenes en los dos grupos que comparamos. El problema aparece cuando los tamaños de las librerías son distintas que es lo habitual. De hecho, suelen ser muy distintos. Una primera idea que no da buenos resultados es dividir los conteos originales por los correspondientes tamaños de las librerías.

Suponemos fijos los tamaños de las librerías. Esto es, aunque suponemos conteos aleatorios la suma de las correspondientes columnas de conteos las asumimos como fijas.[180] Denotamos el mínimo tamaño de las librerías con $m_0 = \min_{j=1,\ldots,n} m_j$ y la librería donde se da este mínimo suponemos que es la j_0. Supongamos que, para la librería j, del total de lecturas elegimos al azar m_0 (correspondiente a la librería con menor tamaño). Esto supone que cada lectura original de esta librería es conservada con probabilidad $\frac{m_0}{m_j}$ y eliminada con probabilidad $1 - \frac{m_0}{m_j}$. Si denotamos por X'_{ij} el número de lecturas que nos quedan después de este proceso de muestreo tendremos que

[180] Una hipótesis necesaria.

$$X'_{ij} | X_{ij} \sim Binomial(X_{ij}, \frac{m_0}{m_j}).$$

Los autores llaman a esto un **muestreo hacia abajo**.[181] Este procedimiento de muestreo nos produce unos nuevos conteos y un nuevo valor del estadístico dado en 10.8 que podemos denotar por T'_i. Notemos que si el tamaño mínimo es mucho menor que el resto de tamaños estamos descartando muchas lecturas.

[181] Down sampling.

Una segunda opción es igualar los tamaños de las librerías a un valor intermedio como la media geométrica $m^* = \left(\prod_{j=1}^{n} m_j\right)^{1/n}$. Generamos conteos aleatorios con

$$X'_{ij}|X_{ij} \sim Poisson\left(\frac{m^*}{m_j}X_{ij}\right). \tag{10.9}$$

Sería el muestreo Poisson. Para evitar empates entre los valores generados se sustituyen estos valores por

$$X'_{ij} + \epsilon_{ij}$$

siendo ϵ_{ij} variables aleatorias independientes y con distribución común uniforme en el intervalo $[0, 0.1]$.[182] Como vemos estamos generando valores por lo que lo que obtenemos también es aleatorio. Supongamos que en un muestreo obtenemos obtenemos $T'_i(b)$. Repetimos este muestreo B veces. Consideramos el estadístico

[182] Un pequeño truco para evitar empates.

$$T^*_i = \frac{1}{B}\sum_{b=1}^{B} T'_i(b). \tag{10.10}$$

Este es el estadístico con el que finalmente se trabaja para contrastar. No conocemos la distribución de este estadístico. Hemos indicado que la región crítica natural, la zona en donde rechazamos la hipótesis nula de que no hay diferencia significativa entre muestras vendría dada por $T^*_i > c$ siendo c un valor a determinar. Dado un valor positivo c consideraremos como significativa a la característica i si $T^*_i > c$. Lo que vamos a hacer es elegir un conjunto de posibles valores para c y **estimaremos** la tasa de hipótesis nulas falsamente rechazadas si utilizamos ese valor de c. El procedimiento[183] es el siguiente:

[183] Ver [129].

1. Calculamos $T^*_1, \ldots, T^*_N$ a partir de nuestros datos.
2. Permutamos S veces las columnas de X manteniendo la asignación de los grupos (las etiquetas de clase) y calculamos $T^*_1(s), \ldots, T^*_N(s)$ utilizando los datos permutados con $s = 1, \ldots, S$.
3. Tomamos una serie de valores para c y calculamos:

$$\hat{V} = \frac{1}{S}\sum_{i=1}^{N}\sum_{s=1}^{S} 1_{|T^*_i(s)|>c} \tag{10.11}$$

que estima la cantidad de hipótesis nulas falsamente rechazadas y

$$\hat{R} = \sum_{i=1}^{N} 1_{|T^*_i|>c} \tag{10.12}$$

que estima la cantidad de hipótesis nulas rechazadas.

4. El valor de FDR asociado a cada valor c es estimado con

$$\widehat{FDR}(c) = \frac{\hat{\pi}_0 \hat{V}}{\hat{R}}. \tag{10.13}$$

donde π_0 es la proporción de hipótesis nulas ciertas.[184] El estimador de π_0 utilizado es

[184] O proporción de características que no son significativamente distintas entre los dos grupos que comparamos.

$$\hat{\pi}_0 = \frac{2\sum_{i=1}^{N} 1_{|T^*_i|\leq q}}{N},$$

siendo q la mediana de $\{|T^*_i(s)| : i = 1, \ldots, N; s = 1, \ldots, S\}$.

10.5.1 SAMseq: PRJNA297664

```
pacman::p_load("SummarizedExperiment","samr")
data(PRJNA297664,package="tamidata")
```

```
PRJNA297664samseq = SAMseq(x = assay(PRJNA297664),
    y = colData(PRJNA297664)[,"treatment"],
    resp.type="Two class unpaired",geneid = rownames(PRJNA297664),
    genenames=rownames(PRJNA297664),nperms = 1000,nresamp = 40,
    fdr.output = 0.20)
```

Podemos ver los resultados con[185]

[185] No mostrado.

```
print(PRJNA297664samseq)
```

Un dibujo con los resultados que podemos ver en figura 10.3

```
plot(PRJNA297664samseq)
```

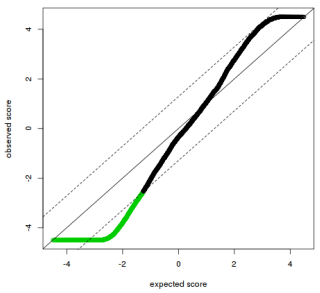


Figura 10.3: Genes significativos con los datos PRJNA297664.

10.6 DESeq2

Este método se propuso en [83].

10.6.1 Método

Como es habitual en el texto denotamos la matriz de conteos aleatoria con $\boldsymbol{X}$ donde el elemento en posición (i, j) es el conteo aleatorio X_{ij} correspondiente al i-ésimo gen y a la j-ésima muestra. Se asume que X_{ij} sigue una distribución binomial negativa (§ 3.8.2) con media μ_{ij} y dispersión α_i. Se supone que la media de este conteo aleatorio la podemos expresar como

$$\mu_{ij} = s_{ij} q_{ij}, \tag{10.14}$$

siendo s_{ij} un factor de normalización que tiene en cuenta las distintas profundidades de secuenciación así como otros posibles sesgos como contenido GC del gen o su longitud.[186] La dependencia de las covariables (variables fenotípicas) se introduce en q_{ij}. En concreto si asumimos para la muestra j un vector de covariables $\boldsymbol{y}_j = (y_{j1}, \ldots, y_{jp})'$ y denotamos un vector de coeficientes (dependiente de la fila o gen) con $\boldsymbol{\beta}_i = (\beta_{i1}, \ldots, \beta_{ip})'$. Se asume el siguiente modelo

$$\log_2 q_{ij} = \boldsymbol{y}_j' \boldsymbol{\beta}_i = \sum_{k=1}^{p} y_{jk} \beta_{ik}.$$

[186] En la propia modelización incorporar el proceso de normalización que, en principio, puede depender del gen y de la muestra. Por defecto se utilizan s_{ij} que no dependen de i y están definidos en § 6.14.3.

[187]

Los **contrastes** vendrán dados por $\boldsymbol{c}'\boldsymbol{\beta}_i$. El error estándar (su desviación típica) del contraste $\boldsymbol{c}'\boldsymbol{\beta}_i$ viene dada por $SE(\boldsymbol{c}'\boldsymbol{\beta}_i) = \sqrt{\boldsymbol{c}'\Sigma_i \boldsymbol{c}}$, donde Σ_i denota la matriz de covarianzas de β_i. Cuando se contraste si estos contrastes[188] son nulos se utiliza la matriz de diseño estándar en donde no se introduce el parámetro adicional que hemos comentado anteriormente. Esta sí es una matriz de rango completo por columnas.

[187] Un detalle importante sobre la codificación de un factor. No se utilizan las habituales variables dummy que codifican uno para una categoría y cero para las demás dejando la media como el valor en una categoría de referencia. Los autores utilizan un parámetro para la media y una variable dummy por cada nivel del factor experimental. Esto ocasiona que

Estimando las dispersiones. Sobre las distintas dispersiones ϕ_i se asume una distribución distribución lognormal.

$$\log \phi_i \sim N(\log \tilde{\phi}(\bar{\mu}_i), \sigma_d^2) \tag{10.15}$$

donde la media es una función $\tilde{\phi}$ que evaluamos en

$$\bar{\mu}_i = \frac{1}{n} \sum_{j=1}^{n} \frac{x_{ij}}{s_{ij}}. \tag{10.16}$$

¿Qué papel desempeña la función $\tilde{\phi}$? Nos describe cómo depende la media de la distribución a priori (sobre el parámetro de dispersion) de la media muestral de los conteos normalizados observados que denotamos $\bar{\mu}_i$. La varianza σ_d^2 modeliza la variabilidad de las dispersiones alrededor de este comportamiento medio que acabamos de indicar. Se utiliza la siguiente función $\tilde{\phi}$.[189]

$$\tilde{\phi}(\bar{\mu}) = \frac{a_1}{\bar{\mu}} + \phi_0. \tag{10.17}$$

En resumen, tenemos tres parámetros en la distribución a priori. [190] Para tener especificada esta distribución a a priori tenemos que estimar a_1, ϕ_0, σ_d^2. Se utiliza un método empírico bayesiano.[191] ¿Cómo los estimamos?

Estimación de las dispersiones ϕ_i's Se empieza ajustando un modelo lineal generalizado con componente aleatoria binomial negativa. Se realiza una primera estimación en donde aplicando el método de los momentos y los conteos estimamos las medias y los parámetros de dispersion. De este modo se tiene una primera estimación de las medias que denotamos $\hat{\mu}_{ij}^0$. Una vez estimadas las medias podemos estimar las dispersiones. Se considera la verosimilitud ajustada de Cox-Reid. Sea $l(\phi)$ la logverosimilitud dada por (eliminamos la referencia a la fila i pero obviamente esta función se considera para cada gen)

$$l(\phi) = \sum_{j=1}^{n} \log f_{NB}(x_j|\mu_j, \phi)$$

donde $f_{NB}(x_j : \mu_j, \phi)$ es la función de probabilidad de la binomial negativa con media μ_j y dispersión ϕ.[192] Denotamos por D la matriz de diseño. La verosimilitud ajustada de Cox-Read viene dada por

$$l_{CR}(\phi|\boldsymbol{\mu}, , \phi) = l(\phi) - \frac{1}{2} \log(det(D'WD)), \tag{10.18}$$

siendo $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)$ y W la matriz de pesos diagonal de los mínimos cuadrados iterativamente reponderados. Dado que la función de enlace es $g(\mu) = \log(\mu)$ y la varianza viene dada por $V(\mu|\phi) = \mu + \phi\mu^2$ entonces el j elemento de la diagonal es

$$w_{jj} = \frac{1}{g'(\mu_j)^2 V(\mu|\phi)} = \frac{1}{\frac{1}{\mu_j} + \phi}.$$

Se maximiza la función dada en 10.18 y obtenemos un estimador que denotamos por ϕ^{gw} para cada gen. Para el i-ésimo gen tenemos $(\bar{\mu}_i, \phi_i^{gw})$.

[189] En [83, pág. 15] hay una interesante discusión sobre la elección de esta función.

[190] Por ser parámetros sobre una distribución de probabilidad para un parámetro realmente hemos de hablar de hiperparámetros en terminología bayesiana.

[191] Los propios datos son utilizados para estimar los hiperparámetros de la distribución a priori.

[192] Definida en 3.21.

Tendencia de la dispersión Pretendemos estimar la función propuesta en 10.17. Se aplica iterativamente un modelo lineal generalizado utilizando la familia de distribuciones gamma. ¿Por qué iterativamente? Porque se van eliminando en cada iteración genes donde el ajuste es malo. ¿En qué sentido? En una iteración dada ajustamos el modelo lineal generalizado. Con el modelo ajustado en esa iteración tendremos una predicción y el valor original (ϕ_i^{gw}). Si el cociente del primero respecto del segundo no está en el intervalo $[10^{-4}, 15]$ es eliminado del estudio. El proceso continúa iterativamente hasta que la suma de los logaritmos de los cocientes de los nuevos coeficientes respecto de los antiguos es menor que 10^{-6}.

Distribución a priori sobre la dispersión Consideremos los residuos logarítmicos dados por

$$\log \phi_i^{gw} - \log \tilde{\phi}(\bar{\mu}_i).$$

La desviación estándar de estos residuos es estimada (evitando observaciones anómalas) utilizando el estimador basado en MAD (mediana de las desviaciones absolutas respecto de la mediana). Es decir con

$$s_{lr} = \frac{1}{\Phi^{-1}(\frac{3}{4})} MAD(\{\log \phi_i^{gw} - \log \tilde{\phi}(\bar{\mu}_i) : i = 1, \ldots, N\}). \quad (10.19)$$

Finalmente, la varianza de la distribución a priori es estimada con

$$\sigma_d^2 = \max\{s_{lr}^2 - \psi_1((n-p)/2), 0.25\}, \quad (10.20)$$

donde ψ_1 es la función trigamma; n, el número de muestras y p el número de covariables por muestra.

El estimador 10.20 no es adecuado cuando los grados de libertad residuales (número de muestras n menos número de parámetros a estimar p) son de tres o inferior. En este caso se propone un método basado en simulación que se puede consultar en [83, pág. 16].

Estimaciones de la dispersión. El estimador final de la dispersión ϕ_i se obtiene

$$\phi_i^{MAP} = argmax_\phi \left(l_{CR}\left(\phi; \bar{\mu}_{i\cdot}^0, x_{i\cdot}\right) + \Lambda_i(\phi) \right) \quad (10.21)$$

siendo

$$\Lambda_i(\phi) = \frac{-(\log \phi - \log \tilde{\phi}(\bar{\mu}_i))^2}{2\sigma_d^2}.$$

Outliers de dispersion. En el punto anterior hemos indicado cómo estimamos la dispersión para cada gen. No obstante, si un gen verifica que

$$\log \phi_i^{gw} > \log \tilde{\phi}(\bar{\mu}_i) + 2s_{lr}$$

entonces consideramos el gen como un outlier en términos de dispersión. Para estos genes utilizamos como estimador de su dispersión no el propuesto en el apartado anterior sino ϕ_i^{gw}. Además estos genes no son utilizados cuando ajustamos $\tilde{\phi}$.

Distribución sobre los coeficientes del modelo Asumimos que

$$\beta_{ir} \sim N(0, \sigma_r^2). \tag{10.22}$$

Para el i-ésimo gen aplicamos el algoritmo (habitual) de los mínimos cuadrados iterativamente reponderados y obtenemos los estimadores máximo verosímiles β_{ir}^{MLE}. Consideramos el cuantil empírico de orden $1-p$ de los $|\beta_{ir}^{MLE}|$ y lo denotamos por $q_{1-p}(|\beta_r|)$. Si considerados $q_N(1-p)$ el cuantil $1-p$ de una normal estándar entonces estimamos con $\sigma_r = \frac{q_{1-p}(|\beta_r|)}{q_N(1-p/2)}$. Se toma por defecto $p = 0.05$.

Estimadores de $\boldsymbol{\beta}_i$.

Se estima $\boldsymbol{\beta}_i$ como el valor que maximiza

$$\sum_{j=1}^{n} f_{NB}(x_{ij}|\mu_j(\boldsymbol{\beta}), \phi_i) + \Lambda(\boldsymbol{\beta}) \tag{10.23}$$

siendo

$$\mu_j(\boldsymbol{\beta}) = s_{ij} e^{\sum_{r=1}^{p} y_{jr}\beta_r},$$

$$\Lambda(\boldsymbol{\beta}) = \sum_{r=1}^{p} \frac{-\beta_r^2}{2\sigma_r^2},$$

y $\phi_i = \phi_i^{MAP}$ excepto para los genes que son outliers de dispersión en donde $\phi_i = \phi_i^{gw}$. La función 10.23 es maximizada utilizando el algoritmo de regresión ridge iterativamente reponderada. En concreto en una iteración dada actualizamos los valores de los coeficientes con

$$\boldsymbol{\beta} \leftarrow (D'WD + \boldsymbol{\lambda} I)^{-1} D'W\boldsymbol{z},$$

donde $\lambda_r = \frac{1}{\sigma_r^2}$ y

$$z_j = \log \frac{\mu_j}{s_j} + \frac{x_j - \mu_j}{\mu_j}$$

siendo $\mu_j(\boldsymbol{\beta}) = s_j e^{\sum_{r=1}^{p} y_{jr}\beta_r}$ donde los valores de β_r son la estimación en la iteración actual.

Test de Wald. En un test de Wald se comparan los estimadores de los coeficientes β_{ir} con sus errores estándar estimados $SE(\beta_{ir}$ que corresponden con el elemento en posición (r, r) de la matriz de covarianzas de $\boldsymbol{\beta}_i$. Esta matriz viene dada por

$$cov(\boldsymbol{\beta}_i) = (D'WD + \boldsymbol{\lambda} I)^{-1}(D'WD)(D'WD + \boldsymbol{\lambda} I)^{-1}.$$

El cociente del estimador con el error estándar tiene aproximadamente una distribución normal estándar bajo la hipótesis nula de que el coeficiente es nulo. Aplicando el test de Wald tendremos p-valores para el test bilateral de que el coeficiente no es nulo. Finalmente estos p-valores pueden ser ajustados mediante algún procedimiento como el método de Benjamini-Hochberg.

En el trabajo se discute la integración de un método de filtrado independiente basado en la media de los conteos normalizados y la adaptación a hipótesis nulas compuestas así como detalles adicionales que no comentamos. En esencia este es el método propuesto.

10.6.2 Paquete

Vamos a analizar los datos tamidata2::PRJNA218851.

```
pacman::p_load(SummarizedExperiment,DESeq2)
```

Leemos los datos.

```
data(PRJNA218851,package="tamidata2")
```

Tenemos la covariable `Stage` que indica el tipo de tejido analizado.

Tenemos tres muestras por individuo. Por tanto tenemos un factor intra sujeto con tres niveles. No vamos a utilizar este diseño. Simplemente vamos a considerar que tenemos tres grupos independientes de muestras.

Empezamos generando el objeto de clase `DESeqDataSet` a partir de nuestro objeto de clase `RangedSummarizedExperiment`.

```
dds =  DESeqDataSet(PRJNA218851, design = ~ Stage)
```

Eliminamos genes con conteos bajos.

```
keep = rowSums(counts(dds)) >= 10
dds = dds[keep,]
```

Fijamos el nivel `control` como categoría de referencia.

```
dds$Stage <- relevel(dds$Stage, ref = "Normal")
```

Podemos hacer el estudio de la expresión diferencial.

```
dds = DESeq(dds)
```

Podemos evaluar el coeficiente correspondiente a la comparación entre el nivel `cancer` y el de referencia `control`.

```
resLFC2 = lfcShrink(dds, coef=2)
```

Ahora evaluamos los tests correspondientes al coeficiente que compara el nivel `metastasis` con el nivel de referencia `control`.

```
resLFC3 = lfcShrink(dds, coef=3)
```

10.7 Material adicional

En [120] tenemos un estudio comparativo de distintos procedimientos para expresión diferencial utilizando distintos paquetes de Bioconductor.

10.8 Ejercicios

Ej. 25 — Para los datos tamidata::PRJNA297664 se pìde:

1. Aplicar el procedimiento de comparación de proporciones utilizando edgeR::binomTest a las muestras 1 y 3.
2. Ajustar los p-valores obtenidos en el punto anterior por el método de Benjamini-Hochberg.
3. Una vez ajustados los p-valores determinar cuantos y cuales tienen un p-valor ajustado menor a 0.01.
4. Repetir los tres pasos anteriores comparando las muestras 2 y 5.
5. ¿Cuántos genes han sido detectados como significativos simultáneamente en los puntos 3 y 4?

10.9 Diseños apareados

En esta sección se considera la situación en que tenemos por individuo dos muestras una en cada condición, tenemos pares de observaciones. Una aproximación bayesiana utilizando la distribución betabinomial se puede encontrar en [**HardcastleKelly2013**].

Capítulo 11

Tamaño muestral y potencia

Los paquetes que vamos a utilizar son

Microarrays [**R-sizepower**],[**R-OCplus**], [**R-ssize**],

RNASeq [**R-RNASeqPower**], [**R-RnaSeqSampleSize**] y sus datos asociados en [**R-RnaSeqSampleSizeData**]. [**R-PROPER**],

Ambos [**R-SSPA**]

ChIP-Seq [**R-CSSP**]

Parte IV

Reducción de dimensión y clasificación

Capítulo 12

Componentes principales

12.1 Introducción

En este tema nos ocupamos de problemas de reducción de dimensión. ¿Qué significa reducir la dimensión? Responder a esta pregunta es obvio si nos fijamos en los datos que tenemos. Trabajando con expresión de genes tenemos tantas filas como genes y tantas columnas como muestras. En resumen miles de filas y decenas o centenares de columnas. En el tema En temas anteriores hemos visto como seleccionar filas, esto es, seleccionar genes es una tarea incluso previa. Hemos de quedarnos con genes que tengan una expresión diferencial si consideramos alguna característica fenotípica o bien con genes que tengan una expresión mínima o bien con genes que tengan un cierto nivel de variación. ¿Qué hacemos con las columnas? O de otro modo: ¿qué hacemos con las muestras? Quizás la respuesta natural sería: si tenemos miles de filas, ¿por qué preocuparse de unas decenas de filas? No es una buena respuesta. Realmente tener 50 o 100 columnas son muchas a la hora de visualizar resultados o bien de aplicar tratamientos estadísticos. En este tema tratamos el tema de cómo reducir el número de columnas o el número de filas.

12.2 Componentes principales

Para ilustrar los conceptos vamos a considerar unos datos sencillos. Tomamos los datos *golub* y nos fijamos en los genes que tienen que ver con "Cyclin" (tienen esta palabra en su nombre). Vamos a considerar las dos primeras muestras, esto es, las dos primeras columnas.

```
data(golub,package="multtest")
sel = grep("Cyclin",golub.gnames[,2])
golub.red = golub[sel,1:2]
```

Los datos aparecen en la figura 12.1.

```
pacman::p_load("ggplot2")
df = data.frame(golub.red)
names(df) = c("sample1","sample2")
```

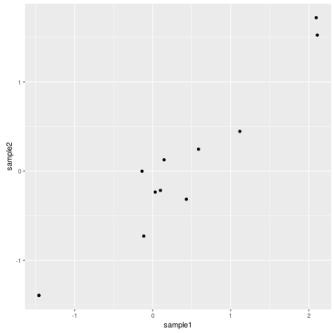


Figura 12.1: Expresión de las dos primeras muestras de los datos golub para aquellos genes que contienen la expresión Cyclin.

```
png(paste0(dirTamiFigures,"PCA2.png"))
ggplot(df,aes(x=sample1,y=sample2)) + geom_point()
dev.off()
```

Cada punto corresponde con uno de los genes seleccionados.

Para la fila i (para el gen i) denotamos las expresiones observadas en las dos muestras como $x_i = (x_{i1}, x_{i1})$. Tenemos n filas y por lo tanto nuestros datos son x_i con $i = 1, \ldots, n$.

Centramos los datos. Esto es, le restamos a cada columna la media de la columna. Para ello, primero calculamos las medias. El vector de medias lo vamos a denotar por $\bar{x} = (\bar{x}_1, \bar{x}_2)$ donde

$$\bar{x}_j = \sum_{i=1}^{n} \frac{x_{ij}}{n}$$

es decir, cada componente es la media de las componentes. En resumen el primer valor es la expresión media en la primera muestra para todos los genes. Podemos calcular fácilmente el vector de medias. Una función específica es la siguiente.

```
medias = colMeans(golub.red)
```

Un código equivalente al anterior con `apply` es

```
medias = apply(golub.red,2,mean)
```

Le restamos a cada columna su media.

```
golub.red = sweep(golub.red,2,medias)
```

En la figura 12.2 reproducimos los datos centrados y mostramos los ejes de coordenadas en rojo.

Hemos trasladado los datos de modo que las medias de cada variable valen cero ahora. Esto es lo que se conoce como centrar los datos. Hemos *centrado los datos*. Podemos comprobar que los nuevos datos tienen una media nula.

```
colMeans(golub.red)

## [1] -3.006854e-17  1.069746e-17
```

Nuestros datos (filas) corresponden a las expresiones correspondientes a los genes. Los datos originales tienen dimensión 2 (dos variables correspondientes a las dos muestras) y supongamos que pretendemos reducir la dimensión a solo una, esto es, representar cada gen mediante un único número. La idea de las componentes principales es considerar una combinación lineal de los valores originales. Es decir, se pretende elegir un vector (de dimensión dos) $a_1 = (a_{11}, a_{12})$ de modo que en lugar de utilizar x_i consideremos (el resumen) $u_i = a_{11}x_{i1} + a_{12}x_{i2}$. ¿Qué a_1 elegimos? La idea es lograr que los valores u_i tengan la mayor variabilidad que se pueda con objeto de no perder información. Mantener la variabilidad original indica que mantenemos la información que los datos originales tienen. En concreto se elige a_1 de modo que maximizamos

$$\frac{1}{n}\sum_{i=1}^{n}(u_i - \bar{u})^2.$$

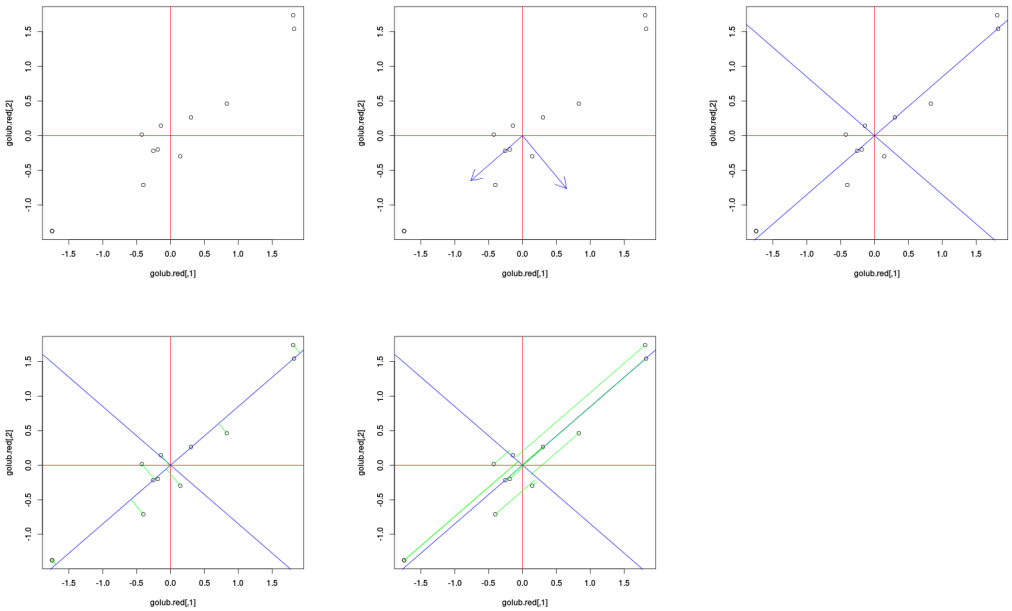


Figura 12.2: a) Centramos los datos golub.red y mostramos el nuevo sistema de coordenadas en rojo. b) Vectores que definen las líneas donde proyectamos. c) Las líneas sobre las que proyectamos. d) Datos golub.red mostrando las líneas sobre las que proyectamos (en azul) para obtener las dos componentes principales y las proyecciones sobre la *primera* componente en verde. e) Datos golub.red mostrando las líneas sobre las que proyectamos (en azul) para obtener las dos componentes principales y las proyecciones sobre la *segunda* componente.

El vector a_1 nos indica la dirección sobre la cual proyectamos los datos originales. Las proyecciones sobre a_1, los valores u_i son la mejor descripción univariante de los datos. La segunda mejor descripción que sea ortogonal a la anterior serían las proyecciones sobre la línea ortogonal a la primera que pasa por el origen de coordenadas. Obtengamos las componentes principales.

```
a.pca = prcomp(golub.red)
```

Los vectores directores de las líneas sobre las que proyectamos aparecen en la figura 12.2. Estos vectores son

```
a.pca$rotation

##              PC1        PC2
## [1,] -0.7619878  0.6475914
## [2,] -0.6475914 -0.7619878
```

Las líneas sobre las que proyectamos aparecen en azul en la figura 12.2 Hemos trasladado los datos de modo que las medias de cada variable valen cero ahora. Esto es lo que se conoce como centrar los datos. Hemos *centrado los datos*. Podemos comprobar que los nuevos datos tienen una media nula.

```
colMeans(golub.red)

## [1] -3.006854e-17  1.069746e-17
```

Nuestros datos (filas) corresponden a las expresiones correspondientes a los genes. Los datos originales tienen dimensión 2 (dos variables correspondientes a las dos muestras) y supongamos que pretendemos reducir la dimensión a solo una, esto es, representar cada gen mediante un único número. La idea de las componentes principales es considerar una combinación lineal de los valores originales. Es decir, se pretende elegir un vector (de dimensión dos) $a_1 = (a_{11}, a_{12})$ de modo que en lugar de utilizar x_i consideremos (el resumen) $u_i = a_{11}x_{i1} + a_{12}x_{i2}$. ¿Qué a_1 elegimos? La idea es lograr que los valores u_i tengan la mayor variabilidad que se pueda con objeto de no perder información. Mantener la variabilidad original indica que mantenemos la información que los datos originales tienen. En concreto se elige a_1 de modo que maximizamos

$$\frac{1}{n}\sum_{i=1}^{n}(u_i - \bar{u})^2.$$

El vector a_1 nos indica la dirección sobre la cual proyectamos los datos originales. Las proyecciones sobre a_1, los valores u_i son la mejor descripción univariante de los datos. La segunda mejor descripción que sea ortogonal a la anterior serían las proyecciones sobre la línea ortogonal a la primera que pasa por el origen de coordenadas. Obtengamos las componentes principales.

```
a.pca = prcomp(golub.red)
```

Los vectores directores de las líneas sobre las que proyectamos aparecen en la figura 12.2(b). Estos vectores son

```
a.pca$rotation
```

```
##             PC1        PC2
## [1,] -0.7619878  0.6475914
## [2,] -0.6475914 -0.7619878
```

Las líneas sobre las que proyectamos aparecen en azul en la figura 12.2(c). Y finalmente podemos ver las proyecciones. En verde mostramos las proyecciones sobre la primera componente (figura 12.2(d)). Y ahora consideremos la proyección sobre la segunda componente. En la figura 12.2(e) la mostramos. Los valores de estas proyecciones los obtenemos con

```
predict(a.pca)
```

```
##               PC1           PC2
##  [1,] -2.50309193 -1.541823e-01
##  [2,]  0.01368602 -2.024163e-01
##  [3,] -2.38702381  3.714339e-03
##  [4,]  0.33489688 -6.847077e-05
##  [5,]  0.76608286  2.806154e-01
##  [6,]  0.27144878  2.899820e-02
##  [7,]  0.31169639 -2.876394e-01
##  [8,]  2.22052303 -8.232084e-02
##  [9,] -0.93221244  1.836866e-01
## [10,] -0.39946389 -7.239549e-03
```

```
## [11,]  0.08293509  3.191733e-01
## [12,]  2.22052303 -8.232084e-02
```

Las desviaciones estándar de la primera y segunda componente principal son las siguientes

```
## [1] 1.4687760 0.1849567
```

Y las varianzas son los cuadrados de las desviaciones estándar.

```
a.pca$sdev^2

## [1] 2.15730290 0.03420899
```

¿Cómo de variables son nuestros datos? Podemos cuantificar el total de la variación de los datos sumando las varianzas de cada una de las dos coordenadas

```
var(golub.red[,1])

## [1] 1.266931

var(golub.red[,2])

## [1] 0.9245807
```

cuya suma es

```
var(golub.red[,1]) + var(golub.red[,2])

## [1] 2.191512
```

Las nuevas coordenadas tienen la misma varianza total.

```
sum(a.pca$sdev^2)

## [1] 2.191512
```

¿Y qué proporción de la varianza es atribuible a la primera componente? ¿Y a la segunda? Podemos dividir la varianza de cada componente por la suma total.

```
variacion.total = sum(a.pca$sdev^2)
a.pca$sdev^2 / variacion.total

## [1] 0.98439023 0.01560977
```

La primera componente explica un 98.44 % de la variación total. ¿Para qué necesitamos utilizar dos números por gen si con uno tenemos esencialmente la misma información.

12.3 Componentes principales de los datos golub

Hemos visto las componentes principales con dos variables (en nuestro caso dos muestras) con efecto de poder ver el significado geométrico de las componentes principales. Vamos a trabajar con el banco

de datos completo: todos los datos golub que tienen 38 muestras (27 de un tipo de leucemia y 11 de otro tipo).

Obtengamos las componentes principales.

```
golub.pca = prcomp(golub,scale=FALSE,center=TRUE)
```

El argumento `center=TRUE` centra los datos restando la media de la columna de modo que las variables tengan medias nulas. El argumento `scale=TRUE` hace que las variables originales sean divididas por su desviación estándar de modo que la varianza (y la desviación estándar) de las nuevas variables sea la unidad.

Diferentes criterios podemos aplicar a la hora de decidir con cuántas componentes nos quedamos.

1. Uno puede ser la proporción total explicada. Fijar un nivel mínimo y quedarnos con el número de componentes necesario para superar este valor mínimo.

2. El segundo puede ser que una componente no puede tener una desviación estándar menor que una de las variables originales. Si hemos escalado cada variable original dividiendo por su desviación estándar entonces la desviación estándar de cada componente ha de ser mayor que uno.

3. Otro criterio puede ser ver en qué momento se produce un descenso de la desviación estándar muy notable. Quedarnos con las componentes previas.

Un resumen de las componentes nos puede indicar con cuántas nos quedamos.

```
summary(golub.pca)
```

```
## Importance of components:
##                            PC1     PC2     PC3
## Standard deviation     5.0436 1.44073 1.11734
## Proportion of Variance 0.6694 0.05462 0.03285
## Cumulative Proportion  0.6694 0.72405 0.75691
##                            PC4     PC5     PC6
## Standard deviation     1.03505 0.85821 0.74399
## Proportion of Variance 0.02819 0.01938 0.01457
## Cumulative Proportion  0.78510 0.80448 0.81905
##                            PC7     PC8     PC9
## Standard deviation     0.72104 0.69232 0.63819
## Proportion of Variance 0.01368 0.01261 0.01072
## Cumulative Proportion  0.83273 0.84534 0.85606
##                           PC10    PC11    PC12
## Standard deviation     0.63630 0.56700 0.55263
## Proportion of Variance 0.01065 0.00846 0.00804
## Cumulative Proportion  0.86672 0.87518 0.88321
##                           PC13    PC14    PC15
## Standard deviation     0.53868 0.52011 0.49568
## Proportion of Variance 0.00764 0.00712 0.00647
## Cumulative Proportion  0.89085 0.89797 0.90443
##                           PC16    PC17    PC18
## Standard deviation     0.48402 0.47719 0.47068
```

```
## Proportion of Variance 0.00617 0.00599 0.00583
## Cumulative Proportion  0.91060 0.91659 0.92242
##                           PC19    PC20    PC21
## Standard deviation     0.45421 0.43795 0.43410
## Proportion of Variance 0.00543 0.00505 0.00496
## Cumulative Proportion  0.92785 0.93290 0.93786
##                           PC22    PC23    PC24
## Standard deviation     0.42475 0.41582 0.40718
## Proportion of Variance 0.00475 0.00455 0.00436
## Cumulative Proportion  0.94260 0.94715 0.95152
##                           PC25   PC26    PC27
## Standard deviation     0.40066 0.3948 0.38731
## Proportion of Variance 0.00422 0.0041 0.00395
## Cumulative Proportion  0.95574 0.9598 0.96379
##                           PC28    PC29    PC30
## Standard deviation     0.38417 0.37882 0.37124
## Proportion of Variance 0.00388 0.00378 0.00363
## Cumulative Proportion  0.96767 0.97145 0.97508
##                           PC31   PC32   PC33
## Standard deviation     0.36957 0.3596 0.3593
## Proportion of Variance 0.00359 0.0034 0.0034
## Cumulative Proportion  0.97867 0.9821 0.9855
##                           PC34    PC35    PC36
## Standard deviation     0.35276 0.34218 0.33228
## Proportion of Variance 0.00327 0.00308 0.00291
## Cumulative Proportion  0.98875 0.99183 0.99473
##                           PC37    PC38
## Standard deviation     0.32572 0.30667
## Proportion of Variance 0.00279 0.00247
## Cumulative Proportion  0.99753 1.00000
```

En este caso podemos aplicar el criterio 1 o 3 ya que no hemos dividido por la desviación estándar.

Si fijamos, por ejemplo un 80% de la variación total a explicar entonces debemos quedarnos con las cinco primeras componentes.

Atendiendo al tercer criterio quizás (muy subjetivo) tampoco sea una mala solución quedarnos con las cinco primeras. Puede ser una buena elección y una solución intermedia. Los nuevos datos los obtenemos con la función *predict*.

```
a = predict(golub.pca)
```

Podemos ver todas las componentes para el primer gen (primera fila).

```
a[1,]
```

```
##          PC1          PC2          PC3
## -7.037559379 -1.611150783 -0.580507718
##          PC4          PC5          PC6
##  0.008740257  0.538496894  0.217863112
##          PC7          PC8          PC9
##  0.095229954 -0.918846037  0.512919401
##         PC10         PC11         PC12
```

```
##  0.863356337 -0.199100855 -0.661871987
##         PC13         PC14         PC15
##  0.098500958  1.167024082 -0.080881207
##         PC16         PC17         PC18
##  0.019311238  0.311828852 -0.734193835
##         PC19         PC20         PC21
##  0.484426444 -0.413971027  0.861064042
##         PC22         PC23         PC24
##  0.412112072 -0.169223915 -0.042496895
##         PC25         PC26         PC27
##  0.392156775 -0.810609198 -0.724090154
##         PC28         PC29         PC30
## -0.022842034 -0.267379502  0.223249663
##         PC31         PC32         PC33
##  0.004501899 -0.066925615 -0.420010671
##         PC34         PC35         PC36
##  0.043025063  0.325939735 -0.095872751
##         PC37         PC38
##  0.451061057  0.873973312
```

Y ahora nos quedamos con las primeras cinco columnas correspondientes con las cinco primeras componentes principales como hemos decidido previamente.

```
a = a[,1:5]
```

Podemos representar, como es habitual, las dos primeras componentes. Lo tenemos en la figura 12.3

```
png(paste0(dirTamiFigures,"PCA28.png"))
df = data.frame(PC1 = a[,1],PC2=a[,2])
ggplot(df,aes(x=PC1,y=PC2)) + geom_point()
dev.off()
```

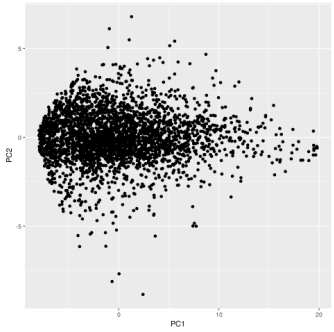


Figura 12.3: Dos primeras componentes principales de los datos golub.

Es interesante observar los valores del vector asociado a la primera componente.

```
golub.pca$rotation[,1]
```

```
##  [1] 0.1715179 0.1690829 0.1650131 0.1726783
##  [5] 0.1659431 0.1668800 0.1686381 0.1602445
##  [9] 0.1648769 0.1687936 0.1653992 0.1694389
## [13] 0.1629073 0.1661268 0.1647691 0.1720833
## [17] 0.1559293 0.1600159 0.1677201 0.1491867
## [21] 0.1272725 0.1620961 0.1643597 0.1652554
## [25] 0.1659262 0.1690494 0.1539691 0.1689052
## [29] 0.1541333 0.1516988 0.1691436 0.1682306
## [33] 0.1452419 0.1675335 0.1638384 0.1508645
## [37] 0.1476137 0.1520465
```

Podemos ver que son coeficientes muy parecidos, todos positivos. Básicamente tenemos la media muestral de todos los niveles de expresión en las 38 muestras. La primera componente es básicamente la media sobre las 38 muestras. ¿Y la segunda componente?

```
golub.pca$rotation[,2]

##  [1]  0.104190349 -0.036887376  0.069108679
##  [4]  0.100701406  0.170952497  0.028349013
##  [7]  0.032390592  0.000505933  0.093593873
## [10]  0.023532773  0.075375878 -0.089380731
## [13]  0.233399832  0.077938472  0.237951078
## [16]  0.184071755  0.078196661  0.041608386
## [19]  0.114629249  0.247148154  0.201580365
## [22] -0.014147623  0.037858911  0.210585781
## [25] -0.044465104  0.122286768  0.021439090
## [28] -0.189278987 -0.174593342 -0.243775839
## [31] -0.165316096 -0.150156242 -0.344034501
## [34] -0.157687744 -0.130649728 -0.277921375
## [37] -0.344828851 -0.222765782
```

Si observamos los coeficientes vemos que las primeros 27 valores son positivos y los 11 últimos son negativos. Además no hay una gran diferencia entre los 27 primeros y tampoco entre los 11 últimos. Básicamente (aunque no exactamente) estamos comparando, para cada gen, la media de los niveles de expresión sobre los datos ALL (leucemia linfoblástica aguda) con la media sobre los datos AML (leucemia mieloide aguda).

12.4 ¿Muestras o genes?

En la sección anterior hemos realizado un análisis de componentes principales *de las muestras*, esto es, la reducción de dimensión suponía resumir, para gen, su perfil de expresión con menos valores. Sin embargo, también podemos realizar un análisis de componentes principales de los genes. En este caso, las observaciones son las muestras y las variables serían las expresiones sobre los genes. Obviamente podemos realizarlo sobre el conjunto de todos los genes o bien después de aplicar alguna selección previa (bien un filtrado no específico, bien utilizando información sobre los propios genes).

Ejemplo 12.1 (Datos golub) *Pretendemos hacer un análisis de componentes principales donde las observaciones son las muestras y para cada muestra tenemos su perfil de expresión sobre todos los genes. Para ello hemos de considerar la matriz traspuesta de la matriz de expresión original con la función* `t()`*. A los datos que obtenemos le aplicamos la función* `prcomp`*.*

```
tgolub.pca = prcomp(t(golub),scale=FALSE,center=TRUE)
```

Veamos el resumen del análisis que acabamos de realizar.

```
summary(tgolub.pca)

## Importance of components:
##                            PC1      PC2     PC3
## Standard deviation     13.0934 10.17462 9.40357
## Proportion of Variance  0.1645  0.09934 0.08485
## Cumulative Proportion   0.1645  0.26385 0.34870
```

```
##                             PC4     PC5     PC6
## Standard deviation     7.9010 6.82616 6.62780
## Proportion of Variance 0.0599 0.04471 0.04215
## Cumulative Proportion  0.4086 0.45332 0.49547
##                             PC7     PC8     PC9
## Standard deviation     6.30435 5.83194 5.79413
## Proportion of Variance 0.03814 0.03264 0.03222
## Cumulative Proportion  0.53361 0.56625 0.59846
##                            PC10    PC11    PC12
## Standard deviation     5.15726 5.01893 4.90719
## Proportion of Variance 0.02552 0.02417 0.02311
## Cumulative Proportion  0.62398 0.64816 0.67126
##                            PC13    PC14    PC15
## Standard deviation     4.72354 4.50857 4.40036
## Proportion of Variance 0.02141 0.01951 0.01858
## Cumulative Proportion  0.69267 0.71218 0.73076
##                            PC16    PC17    PC18
## Standard deviation     4.34750 4.27398 4.12411
## Proportion of Variance 0.01814 0.01753 0.01632
## Cumulative Proportion  0.74890 0.76643 0.78275
##                            PC19    PC20    PC21
## Standard deviation     3.98196 3.94862 3.85795
## Proportion of Variance 0.01522 0.01496 0.01428
## Cumulative Proportion  0.79796 0.81292 0.82721
##                            PC22    PC23    PC24
## Standard deviation     3.77535 3.69767 3.64804
## Proportion of Variance 0.01368 0.01312 0.01277
## Cumulative Proportion  0.84088 0.85400 0.86677
##                            PC25    PC26    PC27
## Standard deviation     3.58436 3.51711 3.49260
## Proportion of Variance 0.01233 0.01187 0.01171
## Cumulative Proportion  0.87910 0.89097 0.90268
##                            PC28    PC29    PC30
## Standard deviation     3.44527 3.37281 3.35604
## Proportion of Variance 0.01139 0.01092 0.01081
## Cumulative Proportion  0.91407 0.92498 0.93579
##                            PC31    PC32    PC33
## Standard deviation     3.26862 3.26192 3.21319
## Proportion of Variance 0.01025 0.01021 0.00991
## Cumulative Proportion  0.94604 0.95625 0.96616
##                            PC34    PC35    PC36
## Standard deviation     3.10703 3.01784 2.95738
## Proportion of Variance 0.00926 0.00874 0.00839
## Cumulative Proportion  0.97543 0.98416 0.99256
##                            PC37      PC38
## Standard deviation     2.78502 7.019e-15
## Proportion of Variance 0.00744 0.000e+00
## Cumulative Proportion  1.00000 1.000e+00
```

En principio, la dimensión de mis datos es de 3051 que es el número de genes con los que estamos trabajando. Sin embargo, vemos que solamente considera 38 posibles componentes principales. ¿Por qué? Pues porque es el rango de la matriz de covarianzas.[1]

[1] Es un comentario incomprensible siguiendo el material anterior. Lo indico

Es interesante representar las componentes principales de las muestras y los genes. En las figuras 12.4 y 12.5 mostramos las dos primeras componentes utilizando la función `plotPCA` *del paquete [86, affycoretools].*

```
library(affycoretools)
png(paste0(dirTamiFigures,"PCA34-a.png"))
affycoretools::plotPCA(golub,legend = FALSE)
dev.off()
png(paste0(dirTamiFigures,"PCA34-b.png"))
affycoretools::plotPCA(t(golub),legend = FALSE)
dev.off()
```

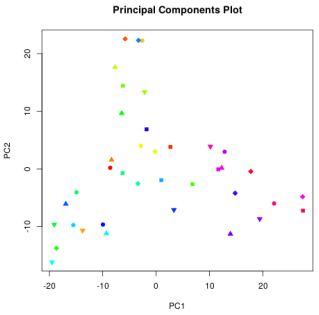


Figura 12.4: Dos primeras componentes principales de los datos golub utilizando como variables las expresiones en los genes.

Cuando realizamos un análisis de los genes sabemos que las observaciones están clasificadas (las muestras son de dos tipos de leucemia). La función `plotPCA` *está preparada para mostrarnos si las componentes nos* reproducen *los grupos que sabemos que previamente tenemos (aquí no hemos realizado ningún análisis cluster previo). En la figura 12.6 mostramos la misma figura 12.4 pero diferenciando el tipo de muestra (según el tipo de leucemia).*

```
tipo = factor(golub.cl+1,levels = 1:2,
              labels = c("ALL","AML"))
png(paste0(dirTamiFigures,"PCA37.png"))
affycoretools::plotPCA(golub,groups = tipo,
                       groupnames = tipo,legend=FALSE)
dev.off()
```

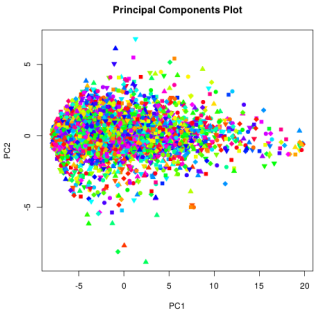


Figura 12.5: Dos primeras componentes principales de los datos golub utilizando como variables las expresiones en las muestras.

Podemos ver cómo la primera componente principal diferencia bastante claramente los dos tipos de leucemia.

12.5 ¿Tipificamos los datos?

Hemos trabajado hasta ahora con los datos centrados (restamos la media muestral a cada variable) pero no hemos tipificado, esto es, no hemos dividido cada variable por su desviación estándar. Si los datos tienen una escala similar esta es la opción razonable. Para matrices de expresión esto es así, estamos midiendo lo mismo en distintas muestras y genes y, sobre el papel, la escala es la misma. Pero: ¿y si no es así? En este caso en lugar de trabajar con las variables originales las tipificamos y realizamos el análisis anterior con estos datos.[193]

Reproducimos (y no mostramos) el análisis de los datos golub pero utilizando datos tipificados. Notemos que la opción fundamental es `scale = TRUE`.

```
golub.pca = prcomp(golub,scale=TRUE,center=TRUE)
summary(golub.pca)
tgolub.pca = prcomp(t(golub),scale=TRUE,center=TRUE)
summary(tgolub.pca)
```

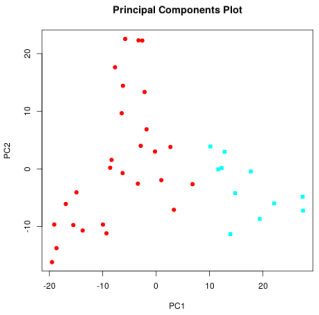


Figura 12.6: Dos primeras componentes principales de los datos golub. donde se aplican las componentes a los genes y diferenciamos la muestra con un color y tipo de punto distintos.

simplemente.

[193] Realmente si no tipificamos y simplementamos centramos los datos estamos realizando un análisis de componentes principales sobre la matriz de covarianzas. Si tipificamos las variables

12.6 Ejemplos

Ejemplo 12.2 (GSE20986) *Cargamos los datos.*

```
pacman::p_load(Biobase)
data(gse20986,package="tamidata")
```

La covariable fenotípica que nos indica el tejido de donde se obtuvo la muestra es

```
pData(gse20986)[,"tissue"]

##  [1] iris      retina    retina    iris
##  [5] retina    iris      choroides choroides
##  [9] choroides huvec     huvec     huvec
## Levels: iris retina choroides huvec
```

En la figura 12.7 vemos representadas las componentes principales de los genes.

```
png(paste0(dirTamiFigures,"PCA44.png"))
plotPCA(gse20986,
     groups = pData(gse20986)[,"tissue"],
     groupnames=c("iris","retina","choroides","huvec"))
dev.off()
```

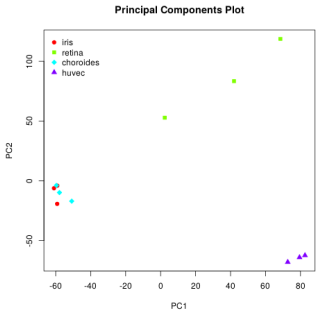


Figura 12.7: Datos GSE20986. Componentes principales de los genes. Se aprecia como el grupo huvec (tejido de vena umbilical) se diferencia claramente de los demás.

Llama la atención la clara diferenciación de las tres muestras correspondientes a huvec. Aparecen dispersas pero separadas de los demás las correspondientes a la retina. Finalmente las muestras de iris y coroides no muestran una diferenciación.

Vemos que son informativas pero: ¿qué proporción de la variación total explican? Realizamos unas componentes principales de las muestras. Notemos que tomamos la matriz de expresión y luego aplicamos la traspuesta ya que de lo contrario las componentes se estarían realizando sobre las muestras.

```
a.pca = prcomp(t(exprs(gse20986)))
summary(a.pca)

## Importance of components:
##                             PC1     PC2      PC3
## Standard deviation     64.0229 58.2172 32.70628
## Proportion of Variance  0.3537  0.2924  0.09229
## Cumulative Proportion   0.3537  0.6461  0.73835
##                             PC4      PC5      PC6
## Standard deviation     27.38208 24.78294 21.58365
## Proportion of Variance  0.06469  0.05299  0.04019
## Cumulative Proportion   0.80304  0.85604  0.89623
##                             PC7     PC8      PC9
## Standard deviation     18.60356 17.9181 15.86741
## Proportion of Variance  0.02986  0.0277  0.02172
## Cumulative Proportion   0.92609  0.9538  0.97551
##                            PC10     PC11
## Standard deviation     13.42905 10.17363
## Proportion of Variance  0.01556  0.00893
```

```
## Cumulative Proportion   0.99107  1.00000
##                             PC12
## Standard deviation     2.117e-13
## Proportion of Variance 0.000e+00
## Cumulative Proportion  1.000e+00
```

Apreciamos que las dos primeras componentes nos explican un 64.61% de la variación total.

En la representación gráfica de la figura 12.7 hemos utilizado la función affycoretools::plotPCA. Quizás no es buena costumbre ligarnos tanto a un tipo de dato como el ExpressionSet. Podemos utilizar [67, ggfortify]. Como antes, hemos de trabajar con la matriz transpuesta y ponemos como una variable adicional de nuestro `data.frame` *la variable que indica el tejido.*

```
df0 = t(exprs(gse20986))
tissue = pData(gse20986)[,"tissue"]
df = data.frame(tissue,df0)
pacman::p_load(ggfortify)
png(paste0(dirTamiFigures,"PCA46b.png"))
autoplot(prcomp(df[,-1]),data=df,colour="tissue")
dev.off()
```

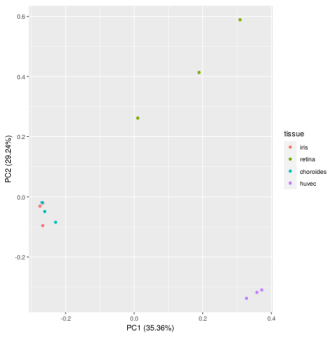


Figura 12.8: Componentes sobre los genes con [67].

Vemos las componentes en la figura 12.8

Ejemplo 12.3 (GSE1397) *Leemos los datos.*

```
data(gse1397,package="tamidata")
```

Tenemos los metadatos fenotípicos con

```
pData(gse1397)
```

Vemos que la variable que nos indica la alteración y el tejido son

```
pData(gse1397)[,'type']
```

```
##  [1] Euploid Euploid Euploid Euploid Euploid
##  [6] Euploid Euploid TS21    TS21    TS21
## [11] TS21    TS21    TS21    TS21
## Levels: Euploid TS21
```

y

```
pData(gse1397)[,'tissue']
```

```
##  [1] Cerebrum   Cerebrum   Cerebrum   Cerebrum
##  [5] Cerebellum Cerebellum Cerebellum Cerebrum
##  [9] Cerebrum   Cerebrum   Cerebrum   Cerebellum
## [13] Cerebellum Cerebellum
## Levels: Cerebrum Cerebellum
```

En la figura 12.9 tenemos el resultado de un análisis de componentes principales considerando los genes como observaciones y diferenciando según el tipo de alteración.

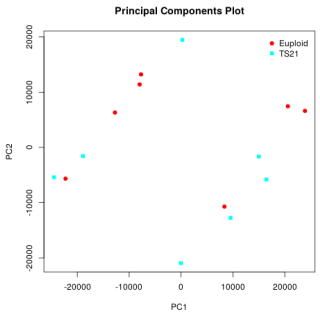


Figura 12.9: Dos primeras componentes principales de datos GSE1397 diferenciando el diagnóstico.

```
png(paste0(dirTamiFigures,"PCA51.png"))
affycoretools::plotPCA(gse1397, groups = pData(gse1397)[,'type'],
        groupnames = levels(pData(gse1397)[,'type']))
dev.off()
```

Las dos primeras componentes vemos como no nos diferencian el tipo de alteración. Sin embargo, cuando consideramos el tejido de donde obtuvimos la muestra se aprecia una mejor diferenciación. De hecho la primera componente diferencia bastante bien los tejidos. Lo podemos ver en la figura 12.10.

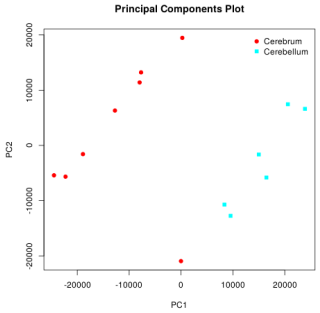


Figura 12.10: Dos primeras componentes principales de datos GSE1397 diferenciando el tejido.

```
library(affycoretools)
png(paste0(dirTamiFigures,"PCA53.png"))
affycoretools::plotPCA(gse1397, groups = pData(gse1397)[,'tissue'],
        groupnames = levels(pData(gse1397)[,'tissue']))
dev.off()
```

12.7 Componentes principales

En esta sección realizamos una presentación más formal del concepto de componente principal. Su nivel no corresponde con el resto del capítulo y se incluye como material complementario.

Cuando tomamos medidas sobre genes, personas, objetos, empresas, unidades experimentales de un modo genérico, se tiende a recoger el máximo de variables posible. En consecuencia tenemos dimensiones del vector de características X grandes.

Una opción consiste en sustituir la observación original, de dimensión d, por k combinaciones lineales de las mismas. Obviamente pretendemos que k sea mucho menor que d. El objetivo es elegir k de modo que expresen una proporción razonable de la dispersión o variación total cuantificada como la traza de la matriz de covarianza muestral, $tr(S)$,

Sea X un vector aleatorio de dimensión d con vector de medias μ y matriz de covarianzas Σ. Sea $T = (t_1, t_2, \ldots, t_d)$ (los t_i indican la i-ésima columna de la matriz) la matriz ortogonal tal que

$$T'\Sigma T = \Lambda = diag(\lambda_1, \ldots, \lambda_d), \tag{12.1}$$

donde $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d \geq 0$ son los valores propios de la matriz Σ. Sea

$$Y = T'(X - \mu). \tag{12.2}$$

Si denotamos la j-ésima componente de Y como Y_j entonces $Y_j = t_j'(X - \mu)$ con $j = 1, \ldots, d$. A la variable Y_j la llamamos la j-ésima **componente principal** de Y. La variable $Z_j = \frac{Y_j}{\sqrt{\lambda_j}}$ es la j-ésima **componente principal estandarizada** de Y.

Estas componentes tienen algunas propiedades de gran interés. Notemos que el vector t_j tiene longitud unitaria y, por lo tanto, Y_j no es más que la proyección ortogonal de $X - \mu$ en la dirección t_j.

Proposición 12.1 *1. Las variables Y_j son incorreladas y $var(Y_j) = \lambda_j$.*

2. Las variables Z_j son incorreladas y con varianza unitaria.

Demostración.

En cuanto al apartado primero tenemos que

$$var(Y) = var(T'(X - \mu) = T' var(X) T = T' \Sigma T = \Lambda.$$

El segundo apartado es directo a partir del primero.

□

Se verifica el siguiente resultado.

Teorema 12.1 *Las componentes principales $Y_j = t_j'(X - \mu)$ con $j = 1, \ldots, d$ tienen las siguientes propiedades:*

1. *Para cualquier vector a_1 de longitud unitaria, $var(a_1'X)$ alcanza su valor máximo λ_1 cuando $a_1 = t_1$.*
2. *Para cualquier vector a_j de longitud unitaria tal que $a_j' t_i = 0$ para $i = 1, \ldots, j-1$, se tiene que $var(a_j'X)$ toma su valor máximo λ_j cuando $a_j = t_j$.*
3. *$\sum_{j=1}^{d} var(Y_j) = \sum_{j=1}^{d} var(X_j) = traza(\Sigma)$.*

La versión muestral de las componentes principales la obtenemos sustituyendo en lo anterior μ y Σ por $\bar{X}$ y $\hat{\Sigma}$ respectivamente. Es importante considerar el estimador de Σ que estamos utilizando (o bien el estimador insesgado donde dividimos por $n-1$ o bien el estimador en donde dividimos por n).

Si denotamos por $\hat{\lambda}_1 \geq \ldots \geq \hat{\lambda}_d$ los valores propios ordenados de $\hat{\Sigma}$ y la matriz $\hat{T} = (\hat{t}_1, \ldots, \hat{t}_d)$ es la matriz tal que cada columna es el correspondiente vector propio entonces tenemos las componentes principales muestrales dadas por $y_j = \hat{T}'(x_i - \bar{x})$. La nueva matriz de datos viene dada por

$$Y' = (y_1, \ldots, y_n) = \hat{T}'(x_1 - \bar{x}, \ldots, x_n - \bar{x}) \tag{12.3}$$

Finalmente, si las variables vienen dadas en unidades muy distintas puede ser conveniente sustituir la matriz de covarianzas (poblacional o muestral) por la correspondiente matriz de correlaciones. De hecho, una de los inconvenientes de las componentes principales como un modo de reducir la dimensión de los datos es precisamente que obtenemos resultados distintos si utilizamos las componentes principales obtenidas a partir de la matriz de covarianzas o bien las componentes principales obtenidas a partir de la matriz de correlaciones.

A partir de las d variables originales podemos obtener hasta d componentes principales. Sin embargo, hemos dicho que pretendemos reducir la dimensión del vector de datos. La pregunta a responder es: ¿con cuántas componentes nos quedamos?

Supongamos que estamos trabajando con la matriz de covarianzas Σ. Hemos de recordar que $var(y_j) = \lambda_j$ y que $\sum_{j=1}^{d} var(x_j) = \sum_{j=1}^{d} var(y_j) = \sum_{j=1}^{d} \lambda_j$. En consecuencia se suelen considerar los siguientes cocientes

$$\frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{d} \lambda_j}, \text{ con } k = 1, \ldots, d,$$

de modo que, cuando para un cierto valor de k, estamos próximos a la unidad nos quedamos con ese valor de k. En la versión muestral

trabajaremos o bien con los valores propios de la matriz de covarianzas muestral o la matriz de correlaciones muestrales.

Una referencia muy interesante sobre componentes principales es [1].

.

Capítulo 13

Análisis cluster

13.1 Introducción

Este tema está dedicado al análisis cluster y su aplicación a datos de expresión de gen. Trataremos lo que en la literatura estadística recibe el nombre de *análisis cluster*[1] o, en mejor castellano, *análisis de conglomerados*. En la literatura de Inteligencia Artificial se utiliza la expresión *clasificación no supervisada*.[2]

¿Qué vamos a clasificar cuando trabajamos con datos de expresión de gen? Nuestra información es la matriz de expresión. Podemos considerar como observaciones a clasificar los genes. En este caso, la observación son los niveles de expresión observados en todas las muestras, el perfil de expresión del gen sobre las muestras analizadas.

También podemos considerar como observaciones a clasificar las muestras. En este caso cada muestra viene descrita por sus expresiones sobre todos los genes. Podemos trabajar con todos los genes o con un subconjunto de genes.

En uno u otro caso tenemos una muestra de observaciones multivariantes de dimensión d. ¿Qué pretendemos hacer con ellos? Encontrar grupos. Muy breve la respuesta pero: ¿qué son grupos? Imaginemos una imagen aérea fija de un patio de un colegio. En esta imagen los datos son las posiciones de los niños. ¿Se agrupan los niños formando grupos o todos están jugando con todos y los grupos son una consecuencia pasajera del juego (delanteros y defensas que aparecen agrupados en un ataque)?

Parece claro y simple el problema. Sí, lo parece. ¿Qué es un grupo? ¿Cómo defino un grupo? ¿Cuántos grupos distingo en los datos? Estamos viendo el efecto del ruido o realmente hay una estructura debajo que la vemos en un entorno con ruido.

¿Qué quiere decir encontrar grupos? Se trata de clasificar las observaciones en grupos de modo que las observaciones de un mismo grupo sean lo más similares que podamos y que los grupos entre sí sean muy distintos. El número de procedimientos que se han propuesto en la literatura es muy grande. La mayor parte de ellos no tienen un modelo probabilístico debajo, no son procedimientos *basados en modelo*. Son métodos que esencialmente utilizan el concepto de *proximidad*. Valoran de distintas formas lo próximos, lo cercanos que están

[1] Por cierto que la palabra cluster no existe en castellano

[2] Por tradición, utilizaremos la expresión de análisis cluster. Es la que, personalmente, prefiero.

los puntos, dentro de un mismo grupo y entre distintos grupos. Es pues, el primer punto a tratar: ¿cómo cuantificamos lo cerca o lejos que están los distintos puntos? En § 13.3 nos ocupamos de este punto. También será necesario, como veremos en el tema, valorar cuando dos conjuntos de puntos son más o menos parecidos, proximos, similares. En la misma sección nos ocupamos de ello. Supongamos que ya hemos clasificado en distintos grupos. ¿Hemos tenido éxito al hacerla? Cuando tenemos un análisis discriminante tenemos una muestra donde sabemos a qué grupo pertenece el individuo y dónde lo hemos clasificado. Esto nos permitía valorar si nuestro procedimiento clasifica bien o no. Aquí no vamos a tener esta referencia que nos da la muestra de entrenamiento. ¿Cómo valorarlo? Un concepto conocido por silueta y debido a Rousseeuw [76] nos va a servir para ello. No es ni tan simple ni tan satisfactorio como en análisis discriminante (como es de esperar si tenemos menos información para trabajar). Lo estudiamos en § 13.8.

Entre los muchos procedimientos de obtener los grupos a partir de los datos, los más utilizados son dos tipos: procedimientos jerárquicos y métodos de particionamiento. De los jerárquicos nos ocupamos en § 13.5. El método de las k-medias y el método de las k-mediodes (el castellano como siempre es muy sufrido pues no existe la palabra) son métodos de particionamiento y los tratamos en § 13.7.

Un texto antiguo pero de un interés enorme en este tema es [127].

13.2 Datos

En esta sección presentamos tres ejemplos sencillos de observaciones a agrupar y que utizamos como ilustración simple de los métodos que proponemos.

13.2.1 Un ejemplo artificial

Son unos datos muy conocidos, los datos Ruspini. Están en el paquete *cluster* [89].[3]

Cargamos el paquete y los datos.

```
pacman::p_load(cluster)
data(ruspini)
```

En la figura 13.1 mostramos estos datos.

```
pacman::p_load(ggplot2)
png(paste0(dirTamiFigures,"Cluster2.png"))
ggplot(ruspini,aes(x=x,y=y))+geom_point()
dev.off()

## pdf
##   2
```

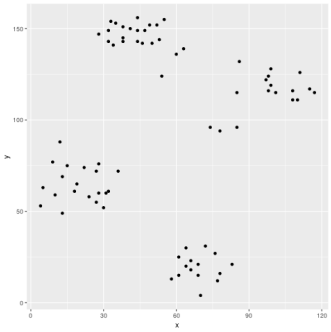


Figura 13.1: Datos ruspini

Son datos bivariantes. Visualmente vemos cómo se agrupan los puntos. Parece (muy) claro que podemos distinguir cuatro grupos.

[3] En este tema el paquete cluster es, sin duda, el fundamental.

13.2.2 Un ejemplo con muestras

Un ejemplo con los datos golub. Empezamos cargando los datos.

```
data(golub,package="multtest")
```

Previamente hemos visto que los valores de expresión de los genes "CCND3 Cyclin D3" y "Zyxin" permiten diferenciar entre ALL y AML. Localicemos las expresiones correspondientes a estos genes.

```
grep("CCND3 Cyclin D3",golub.gnames[,2])

## [1] 1042

grep("Zyxin",golub.gnames[,2])

## [1] 2124
```

Los datos aparecen en estas filas. Por lo tanto podemos construir la matriz de datos correspondiente.

```
cz.data = data.frame(golub[1042,],golub[2124,])
names(cz.data) = c("CCND3_Cyclin_D3","Zyxin")
```

Este será un segundo ejemplo para analizar. Los datos aparecen en la figura 13.2.

```
png(paste0(dirTamiFigures,"Cluster6b.png"))
ggplot(cz.data,aes(x=CCND3_Cyclin_D3,y=Zyxin))+geom_point()
dev.off()

## pdf
##   2
```

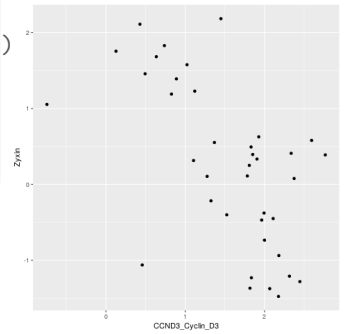


Figura 13.2: Datos cz.data

En este caso las observaciones corresponden a las muestras y las variables son los niveles de expresión de dos genes. ¿Hay grupos? Esto no son datos artificiales como los de Ruspini y ya no es tan claro.

13.2.3 Un ejemplo con genes

Consideremos (otra vez) los datos *golub*.

```
data(golub,package="multtest")
```

Para cada gen vamos a considerar la expresión media sobre los casos ALL

```
x = apply(golub[,golub.cl == 0],1,mean)
```

y la media sobre los casos AML.

```
y = apply(golub[,golub.cl == 1],1,mean)
```

Y vamos a limitarnos a un conjunto de genes. En concreto vamos a considerar los genes que se relacionan con el término biológico "Cyclin". Determinamos qué filas ocupan.

```
sel = grep("Cyclin",golub.gnames[,2])
```

La función *grep* busca la palabra "Cyclin" en cada uno de los nombres que le pasamos. Y los nombres de estos genes son

```
golub.gnames[sel,2]

##  [1] "CCND2 Cyclin D2"
##  [2] "CDK2 Cyclin-dependent kinase 2"
##  [3] "CCND3 Cyclin D3"
##  [4] "CDKN1A Cyclin-dependent kinase inhibitor 1A (p21, Cip1)"
##  [5] "CCNH Cyclin H"
##  [6] "Cyclin-dependent kinase 4 (CDK4) gene"
##  [7] "Cyclin G2 mRNA"
##  [8] "Cyclin A1 mRNA"
##  [9] "Cyclin-selective ubiquitin carrier protein mRNA"
## [10] "CDK6 Cyclin-dependent kinase 6"
## [11] "Cyclin G1 mRNA"
## [12] "CCNF Cyclin F"
```

Ahora nos limitamos a considerar las expresiones medias para los casos ALL y los casos AML para estos genes.

```
cyclin.data = data.frame(x[sel],y[sel])
names(cyclin.data) = c("meanALL","meanAML")
```

En la figura 13.3 mostramos los datos.

```
png(paste0(dirTamiFigures,"Cluster14.png"))
ggplot(cyclin.data,aes(x=meanALL,y=meanAML))+geom_point()
dev.off()

## pdf
##   2
```

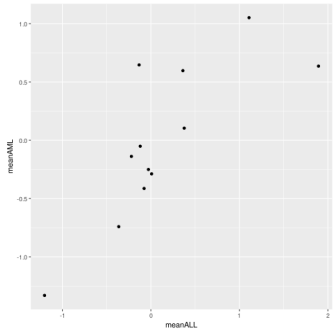


Figura 13.3: Datos cyclin.data.

No parece, al menos visualmente, que tengamos muchos grupos. De eso se trata. De saber si hay grupos y encontrarlos.

13.3 Disimilaridades

En lo que sigue denotaremos los datos a agrupar con x_i con $i = 1, \ldots n$ siendo $x_i \in \mathbb{R}^d$. Todos los ejemplos que hemos visto en la sección anterior eran (con objeto de poder representarlos) datos bidimensionales, $d = 2$.

13.3.1 Distancias euclídea y de Manhattan

Empezamos tratando el problema de cuantificar el grado de proximidad, de similaridad entre dos puntos en el espacio de dimensión d. Tradicionalmente este tema en Matemáticas se ha formalizado a través del concepto de *distancia* o *métrica*. Una métrica es una función que a cada par de puntos $x, y \in \mathbb{R}^d$ le asocia un valor positivo de modo que cuando mayor es más *distantes* son, más alejados están. Como siempre la formalización matemática de un concepto intuitivo ha de

ser prudente y pedir que se verifiquen ciertos axiomas que resulten razonables y generalmente admisibles. Las distancias más utilizadas en análisis cluster son la distancia euclídea y la distancia de Manhattan. Para dos vectores x e y (en $\mathbb{R}^d$) entonces la distancia euclídea se define como

$$d_E(x,y) = \sqrt{\sum_{k=1}^{d}(x_k - y_k)^2}, \tag{13.1}$$

con $x, y \in \mathbb{R}^d$. La distancia de Manhattan viene dada por

$$d_M(x,y) = \sum_{k=1}^{d}|x_k - y_k|. \tag{13.2}$$

Las distancias euclídea y de Manhattan son adecuadas cuando trabajamos con variables continuas y que además estén en una misma escala. Son dos ejemplos de distancias o métricas.[4] En lo que sigue en lugar de distancia utilizaremos el término *medida de disimilaridad* indicando que no necesariamente ha de ser distancia. Es una visión menos restrictiva a la hora de definir que dos genes están cerca.

En lo que sigue agrupamos bien genes bien muestras. Si esta clasificación no supervisada la basamos en una distancia lo que buscamos es encontrar grupos de genes que tienen perfiles de expresión similares. En definitiva que se comportan de un modo parecido en el experimento para todas las muestras. Si agrupamos muestras lo que buscamos son muestras que, para los genes considerados, tienen unas expresiones similares. Por ejemplo, si las muestras corresponden a un cierto tipo de cáncer entonces los grupos que podemos encontrar serían subtipos de cáncer. En resumen genes o muestras los consideramos similares cuando sus expresiones lo son. Además interpretamos la similaridad como una métrica. El concepto de distancia o métrica (del cual hemos visto dos ejemplos) es limitado en muchas aplicaciones. En ocasiones podemos interpretar que dos genes están *próximos* cuando sus expresiones están *relacionadas* de alguna forma, en resumen, cuando están *corregulados*. En este caso podemos usar para definir la disimilaridad a partir de una cuantificación de la dependencia entre los genes.

13.3.2 Disimilaridades utilizando coeficientes de correlación

Consideremos dos genes con perfiles $x = (x_1, \ldots, x_n)$ e $y = (y_1, \ldots, y_n)$. En primer lugar pretendemos cuantificar el grado de dependencia entre los pares (x_i, y_i) para $i = 1, \ldots, n$. Si nos limitamos a la posible dependencia lineal entre estos pares entonces el coeficiente de correlación de Pearson es la medida genéricamente aceptada. En § 8 tenemos su definición y algunas propiedades básicas. Se define como

$$r_{x,y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x}_n)(y_i - \bar{y}_n)}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x}_n)^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y}_n)^2}}.$$

[4] En concreto la función d definida en el espacio producto $\mathbb{R}^d \times \mathbb{R}^d$ se dice que es una métrica si verifica:

No negativa $d(x,y) \geq 0$.

Un punto dista 0 de sí mismo $d(x,x) = 0$.

Simetría $d(x,y) = d(y,x)$.

Desigualdad triangular $d(x,z) \leq d(x,y) + d(y,z)$, para todo $x, y, z \in \mathbb{R}^d$.

Lo fundamental es que varía entre -1 y 1. Cuando toma valores próximos a cero indica independencia entre los pares y cuando toma valor próximo a 1 o a -1 indica una dependencia lineal extrema. De hecho si vale 1 entonces existen constantes a y b tales que $y_1 = ax_i + b$ con $a > 0$. Si vale -1 entonces existen constantes a y b tales que $y_1 = ax_i + b$ con $a < 0$.

Supongamos que cuando evaluamos si dos genes están *próximos* o están *alejados* lo que estamos intentando expresar es que están corregulados. Esto indicaría que una mayor expresión de uno en una muestra se corresponde con una mayor expresión en la misma muestra. Si, además, la asociación es de tipo lineal esto significaría que el perfil de expresión de gen se podría obtener como una función lineal de la expresión del otro gen. En resumen su coeficiente de correlación de Pearson (§ 8) debiera de ser próximo a uno. También sería razonable entender como próximos aquellos genes donde cuando uno tiene un valor alto de expresión el otro lo tiene bajo. Es decir, finalmente tienen una fuerte relación lineal pero de tipo inverso.

A partir del coeficiente de correlación podemos definir distintas medidas de disimilaridad. Algunas opciones que se han propuesto son las siguientes:

1. $d(x, y) = 1 - r_{x,y}$,
2. $d(x, y) = (1 - r_{x,y})/2$,
3. $d(x, y) = 1 - |r_{x,y}|$,
4. $d(x, y) = \sqrt{1 - r^2_{x,y}}$.

[5] De un modo similar podemos definir disimilaridades sustituyendo el coeficiente de correlación de Pearson con el coeficiente de correlación de Spearman.

13.3.3 Matriz de disimilaridades

Tenemos una serie de conjunto de puntos a agrupar: $\{x_1, \ldots, x_n\}$. Una vez elegida una disimilaridad construiremos una matriz que en la fila i, columna j tiene la disimilaridad entre el punto x_i y el punto x_j. Denotaremos indistintamente $d(x_i, x_j)$ o d_{ij}. Por tanto, la matriz de disimilaridades es

$$D = [d(x_i, x_j)]_{i,j=1,\ldots,n} = [d_{ij}]_{i,j=1,\ldots,n}.$$

¿Cómo calculamos las matrices de disimilaridades que acabamos de proponer? La función *dist* de [109] es una buena opción cuando utilizamos las distancias euclídea o de Manhattan.

Un detalle es importante. Si la disimilaridad que estamos considerando es simétrica, esto es, si la disimilaridad d verifica

$$d(x_i, x_j) = d(x_j, x_i),$$

entonces la matriz de disimilaridades

$$D = [d(x_i, x_j)]_{i,j=1,\ldots,n}$$

[5] En http://research.stowers-institute.org/efg/R/Visualization/cor-cluster/ podemos encontrar un breve estudio comparativo del funcionamiento de estas disimilaridades en clustering jerárquico.

es simétrica y estamos almacenando información redundante. Además hemos de considerar que

$$d(x_i, x_i) = 0,$$

es decir, la disimilaridad de un perfil de expresión consigo mismo es nula y por lo tanto, en la matriz de disimilaridades, la diagonal de la matriz es nula. Sabemos que trabajamos habitualmente con un número de genes muy grande. La matriz de disimilaridades total sería de tamaño $N \times N$ y por ello, es más que conveniente eliminar redundancias. Veamos cómo esto lo hace de un modo automático la función `dist`.

Vamos a calcular la matriz de distancias con la función *dist* para los datos *cz.data* propuestos en § 13.2.2.

```
cz.dist.euclidea = dist(cz.data,method="euclidian")
```

¿De qué clase es `cz.dist.euclidea`.

```
class(cz.dist.euclidea)

## [1] "dist"
```

La distancia de Manhattan la podemos obtener con

```
cz.dist.manhattan = dist(cz.data,method="manhattan")
```

```
data(golub,package="multtest")
golub.cor1 = 1-cor(t(golub))
golub.cor2 = (1-cor(t(golub)))/2
golub.cor3 = 1-abs(cor(t(golub)))
golub.cor4 = sqrt(1-cor(t(golub))^2)
```

En este caso `cz.dist` sí es una matriz completa (con todas sus filas y columnas). La vamos a transformar a un objeto de clase `dist`.

```
golub.cor1 = as.dist(golub.cor1)
golub.cor2 = as.dist(golub.cor2)
golub.cor3 = as.dist(golub.cor3)
golub.cor4 = as.dist(golub.cor4)
```

Con cada una de estas matrices podemos cuantificar la disimilaridad que hay entre los elementos originales de la muestra. Algunos de los procedimientos de agrupamiento que vamos a considerar en lo que sigue no necesitan conocer los datos originales. Pueden aplicarse con solo conocer esta matriz de disimilaridades. Otros no. Otros utilizan los datos a lo largo de las distintas etapas de aplicación del procedimiento.

13.4 Disimilaridades entre grupos de observaciones

En algunos procedimientos de agrupamiento (en particular, los jerárquicos) vamos a necesitar calcular disimilaridades entre conjuntos

disjuntos de las observaciones originales. Estas disimilaridades las podemos calcular a partir de las disimilaridades originales entre puntos. Supongamos que tenemos un banco de datos con n individuos cuyos índices son $\{1, \ldots, n\}$. Sean A y B dos subconjuntos disjuntos del conjunto de índices de la muestra $\{1, \ldots, n\}$, esto es, dos subconjuntos de observaciones disjuntos. ¿Cómo podemos definir una disimilaridad entre A y B partiendo de las disimilaridades entre los datos individuales? Se han propuesto muchos procedimientos. Si denotamos la disimilaridad entre A y B como $d(A, B)$ entonces las disimilaridades más habitualmente utilizadas son las siguientes:

Enlace simple La disimilaridad entre los dos grupos es el mínimo de las disimilaridades entre las observaciones de uno y de otro. Tomamos la disimilaridad de los objetos que más se parecen en uno y otro grupo.

$$d(A, B) = \min_{a \in A, b \in B} d(a, b)$$

Enlace completo Ahora tomamos como disimilaridad entre los grupos como el máximo de las disimilaridades, en definitiva, la disimilaridad entre los objetos más alejados o más distintos.

$$d(A, B) = \max_{a \in A, b \in B} d(a, b)$$

Promedio La disimilaridad es el promedio de las disimilaridades entre todos los posibles pares.

$$d(A, B) = \frac{1}{|A| \times |B|} \sum_{a \in A, b \in B} d(a, b)$$

donde $|A|$ es el cardinal del conjunto A.

Es importante notar que solamente necesitamos conocer las disimilaridades entre los individuos para poder calcular las disimilaridades entre grupos de individuos.

En la siguiente sección nos vamos a ocupar de los métodos jerárquicos en los cuales es fundamental el procedimiento que elijamos para calcular distintas entre grupos.

13.5 Cluster jerárquico

La idea de estos procedimientos es construir una jerarquía de particiones del conjunto de índices.

Sea $\{1, \ldots, n\}$ el conjunto de índices que indexan las distintas observaciones. Supongamos que $\{C_1, \ldots, C_r\}$ es una partición de este conjunto de índices:

- $C_i \subset \{1, \ldots, n\}$; son disjuntos dos a dos, $C_i \cap C_j = \emptyset$ si $i \neq j$ con $i, j = 1, \ldots, n$ y
- $\cup_{i=}^{r} C_i = \{1, \ldots, n\}$.

Dada una partición del conjunto de índices podemos calcular la matriz $r \times r$ que en la posición (i, j) tiene la disimilaridad entre el conjunto C_i y C_j, $d(C_i, C_j)$ según alguno de los procedimientos antes indicados.

Veamos los procedimientos jerárquicos aglomerativos. En estos procedimientos vamos a iniciar el agrupamiento con la partición: $C_i = \{i\}$ con $i = 1, \ldots, n$, es decir, cada grupo es un individuo. En cada iteración vamos agrupando el par de conjuntos (elementos de la partición que tengamos en esa iteración) que estén *más próximos* según la disimilaridad entre grupos que estemos utilizando. El proceso continúa hasta que tengamos un único grupo.

Un esquema algorítmico del procedimiento indicado puede ser el siguiente:

Paso 0 Tenemos grupos unitarios formados por cada una de las observaciones. Tenemos pues una partición inicial $C_i = \{i\}$ con $i = 1, \ldots, n$. En un principio, cada dato es un grupo.

Paso 1 Calculamos las disimilaridades entre los elementos de la partición. Para ello utilizamos cualquiera de los procedimientos antes indicados.

Paso 2 Agrupamos los dos conjuntos de la partición más próximos y dejamos los demás conjuntos igual. Tenemos ahora C_i con $i = 1, \ldots, k$.

Paso 3 Si tenemos un solo conjunto en la partición paramos el procedimiento.

Paso 4 Volvemos al paso 1.

Hay una representación gráfica muy utilizada para describir los resultados de un cluster jerárquico aglomerativo como el que acabamos de describir. Esta representación tiene el nombre de **dendograma**. En el dendograma se va mostrando a qué valor de la medida de disimilaridad se produce la unión de los grupos y simultáneamente qué grupos se están uniendo para esa disimilaridad. También nos permite una valoración rápida de cuántos grupos puede haber en el banco de datos. Simplemente trazando una linea horizontal a la altura en que tengamos el número de grupos que *pensamos* que puede haber.

Ejemplo 13.1 (ruspini) *Consideremos los datos ruspini. Apliquemos un cluster jerárquico aglomerativo utilizando como disimilaridad entre grupos el promedio de las disimilaridades y como medida de disimilaridad la distancia euclídea.*

```
pacman::p_load(cluster)
 ruspini.ag = agnes(ruspini,metric = "euclidean",method="average")
```

Una opción alternativa con base::hclust al código anterior es

```
dd = dist(ruspini,method="euclidian")
ruspini.ag = hclust(dd,method="average")
```

Representamos el dendograma en la figura 13.4.

```
png(paste0(dirTamiFigures,"Cluster23.png"))
ggdendro::ggdendrogram(ruspini.ag)
dev.off()

## pdf
##   2
```

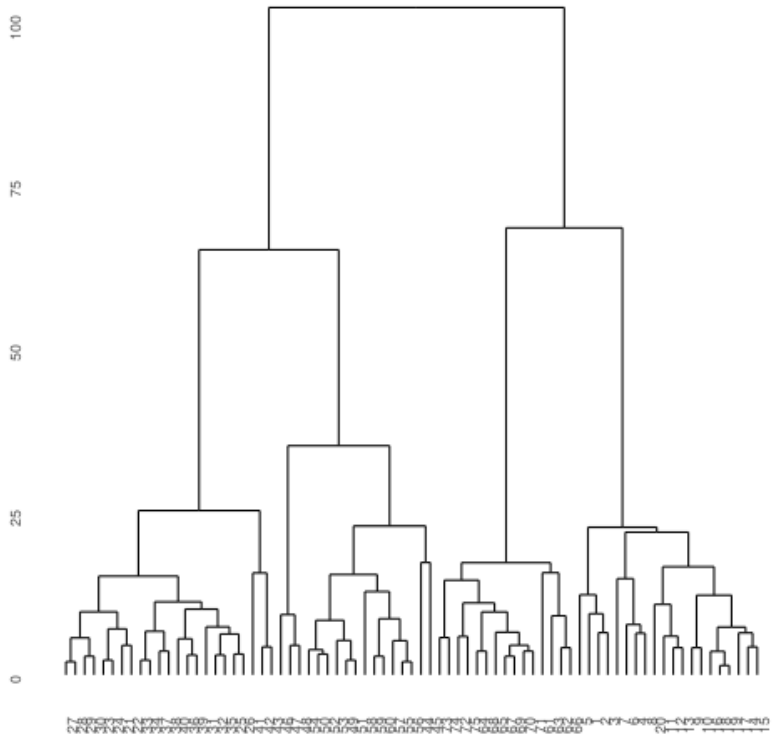


Figura 13.4: Dendograma para un cluster jerárquico aplicado a los datos ruspini con métrica euclídea y el promedio para la disimilaridad entre grupos.

Supongamos que decidimos quedarnos con cuatro grupos. Las clasificaciones de los datos son las siguientes donde la etiqueta que se asigna a cada grupo es arbitraria.

```
cutree(ruspini.ag,4)
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
##  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
## 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
##  1  1  1  1  2  2  2  2  2  2  2  2  2  2  2  2
## 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
##  2  2  2  2  2  2  2  2  2  2  2  3  3  3  3  3
## 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
##  3  3  3  3  3  3  3  3  3  3  3  3  4  4  4  4
## 65 66 67 68 69 70 71 72 73 74 75
##  4  4  4  4  4  4  4  4  4  4  4
```

Y ahora podemos representar los datos de modo que ponemos en un mismo color los datos que hemos clasificado en un grupo. Lo mostramos en la figura 13.5.

```
png(paste0(dirTamiFigures,"Cluster26.png"))
p = ggplot(ruspini,aes(x=x,y=y,color=cutree(ruspini.ag,4)))
p + geom_point(shape=cutree(ruspini.ag,4))
dev.off()
```

```
## pdf
##   2
```

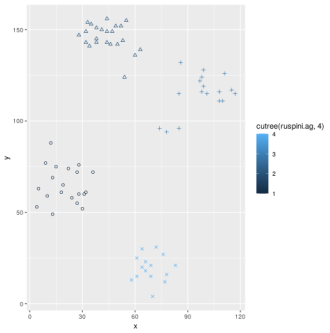


Figura 13.5: Clasificación obtenida para los datos ruspini utilizando un cluster jerárquico con distancia euclídea, promedio entre grupos y cuatro grupos.

Ejemplo 13.2 (cz.data) *Apliquemos un cluster jerárquico aglomerativo utilizando como disimilaridad entre grupos el enlace simple y como distancia la de Manhattan.*

```
##cz.ag =  agnes(cz.data,metric = "euclidean",method="single")
dd = dist(cz.data,method="euclidian")
cz.ag = hclust(dd,method="single")
```

La figura 13.6 muestra el dendograma que obtenemos.

```
pacman::p_load(ggdendro)
png(paste0(dirTamiFigures,"Cluster29.png"))
ggdendro::ggdendrogram(cz.ag , rotate = FALSE, size = 2)
dev.off()
```

```
## pdf
##   2
```

Supongamos que nos quedamos cinco grupos. Las clasificaciones son:

```
cutree(cz.ag,5)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1
## [24] 1 1 1 1 3 4 3 3 3 3 3 5 3 3 3
```

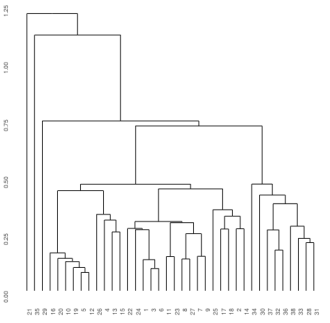


Figura 13.6: Dendograma para datos cz.data utilizando distancia euclídea y enlace simple.

En la figura 13.7 tenemos los resultados de la clasificación.

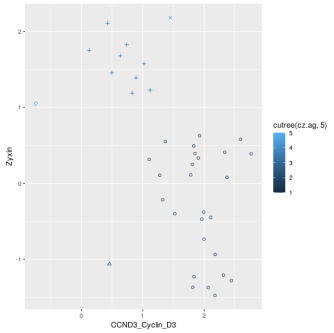


Figura 13.7: Clasificación obtenida para los datos cz.data utilizando un cluster jerárquico cuando cortamos el arbol en cinco grupos. Se utilizó distancia euclídea y enlace simple.

```
png(paste0(dirTamiFigures,"Cluster32.png"))
p = ggplot(cz.data,aes(x=CCND3_Cyclin_D3,y=Zyxin,color=cutree(cz.ag,5)))
p + geom_point(shape=cutree(cz.ag,5))
dev.off()

## pdf
##   2
```

13.6 Distancia cofenética

Una vez hemos construido un clustering jerárquico podemos asociarle una distancia conocida como *distancia cofenética*. Dadas dos observaciones (en nuestro caso dos perfiles de expresión) consideramos el momento en que ambas observaciones han sido agrupadas en un mismo grupo. La distancia cofenética entre estas observaciones es la distancia entre los dos grupos a los que previamente pertenecían estas observaciones (una en cada uno de ellos) en el momento en que se unen. Se calcula con la función `cophenetic` del paquete [109, stats].

Ejemplo 13.3 (Distancia cofenética con cyclin.ag) *Continuamos con el jerárquico aglomerativo aplicado a los datos* cyclin.data. *Repetimos el análisis cluster calculando previamente la matriz de disimilaridades. Empezamos calculando la matriz de distancias.*

```
cyclin.manhattan = daisy(cyclin.data,metric = "manhattan")
```

Y ahora el cluster jerárquico (observemos que no le damos los datos sino que le pasamos directamente la matriz de disimilaridades).

```
##cyclin.ag = agnes(cyclin.manhattan,diss = TRUE)
cyclin.ag = hclust(cyclin.manhattan,method="average")
```

Calculamos la distancia cofenética asociada al cluster jerárquico que acabamos de realizar.

```
cyclin.cofe = cophenetic(cyclin.ag)
```

Y determinamos el coeficiente de correlación de Pearson.

```
cor(cyclin.manhattan,cyclin.cofe)

## [1] 0.8501727
```

El coeficiente de correlación observado 0.8501727 es un valor grande. Tenemos pues una buena descripción del modo en que los datos se agrupan.

Las distancias cofenéticas tienen sus problemas. Es una manera *natural* de medir disimilaridad entre los datos a partir del modo en que estos van agrupándose. Sin embargo, notemos que muchos pares de observaciones tienen la misma disimilaridad. Esto es lógico, cuando

unimos dos grupos grandes la distancia cofenética entre cada observación de un grupo y del otro grupo tienen la misma distancia cofenética. Se suele calcular el coeficiente de correlación entre la matriz de disimilaridad original y la distancia cofenética. Si el valor es grande indica que el dendograma es una buena descripción de los datos.

Un estudio mucho más amplio de lo visto en esta sección puede verse en [127, capítulo 5].

13.7 Métodos de particionamiento

Suponemos ahora que tenemos una idea de cuántos grupos hay. Posiblemente hemos realizado un análisis jerárquico previo con todos los datos o, si eran muchos, con una selección aleatoria de los datos. Tenemos pues una idea de cuántos grupos tendremos que considerar. Obviamente podemos luego evaluar los resultados modificando el número de grupos. En principio, vamos a suponer que fijamos el número de grupos a considerar. Suponemos pues que *sabemos* el número de grupos y lo denotamos por k.

13.7.1 Método de las k-medias

El primer procedimiento que vamos a ver es el método de las *k-medias* (que por alguna extraña razón en la literatura de Inteligencia Artificial se le llama de las *c-medias* lo que demuestra que cada persona copia a sus amigos o, simplemente, conocidos). Supongamos que tenemos $C_1, \ldots, C_k$ una partición de $\{1, \ldots, n\}$. Un modo bastante natural de valorar la calidad del agrupamiento que la partición nos indica sería simplemente considerar la siguiente función.

$$\sum_{i=1}^{k} \sum_{j \in C_i} d_E(x_j, \bar{x}_{C_i})^2, \tag{13.3}$$

donde d_E denota aquí la distancia euclídea y

$$\bar{x}_{C_i} = \frac{1}{|C_i|} \sum_{j \in C_i} x_j, \tag{13.4}$$

es el vector de medias del grupo cuyos índices están en C_i. Una partición será tanto mejor cuanto menor sea el valor de la función dada en 13.3. El procedimiento de agrupamiento de las k-medias simplemente se basa en elegir como partición de los datos aquella que nos da el *mínimo* de la función objetivo considerada en ecuación 13.3. Notemos que en muchos textos se hablan del algoritmo de las k-medias y se identifica con un procedimiento concreto para encontrar el mínimo de la función. Aquí entendemos el procedimiento como la minimización de la función objetivo. De hecho, *R* ofrece hasta cuatro posibles procedimientos de los muchos que cabe proponer. Hay que diferenciar claramente el procedimiento del método de aplicación del mismo, del método de obtención de dicho mínimo.

Es importante darnos cuenta de que el procedimiento que acabamos de ver está basado en la utilización de la distancia euclídea y en que, dado un grupo, podemos calcular el vector de medias y esto solo lo podemos hacer si todas las variables son cuantitativas.

Vamos a aplicar el método de las k-medias en los tres ejemplos utilizando el número de grupos que hemos utilizado previamente.

Ejemplo 13.4 (ruspini) *Aplicamos el k-medias.*

```
ruspini.km = kmeans(ruspini,4)
```

La clasificación viene dada por

```
ruspini.km$cluster
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
##  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
## 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
##  1  1  1  1  3  3  3  3  3  3  3  3  3  3  3  3
## 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
##  3  3  3  3  3  3  3  3  3  3  3  4  4  4  4  4
## 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
##  4  4  4  4  4  4  4  4  4  4  4  4  2  2  2  2
## 65 66 67 68 69 70 71 72 73 74 75
##  2  2  2  2  2  2  2  2  2  2  2
```

La clasificación obtenida se corresponde con la que vemos en la figura 13.5.

Ejemplo 13.5 (Método de las k-medias y datos cz.data) *Empezamos con el k-medias.*

```
cz.km = kmeans(cz.data,2)
```

Las clasificaciones son

```
cz.km$cluster
```

```
##  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [24] 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1
```

Y representamos los resultados.

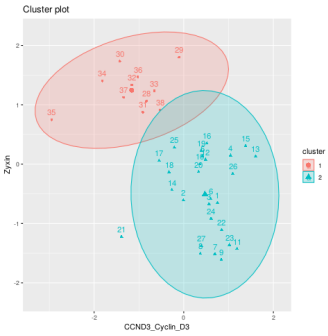


Figura 13.8: Clasificación obtenida utilizando el método de las k-medias aplicada a los datos cz.data.

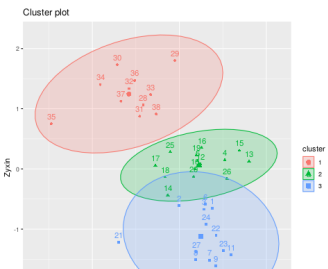
```
pacman::p_load(factoextra)
png(paste0(dirTamiFigures,"Cluster50.png"))
fviz_cluster(cz.km,cz.data, ellipse.type = "norm")
dev.off()

## pdf
##   2
```

Supongamos que probamos con tres grupos.

```
cz.km = kmeans(cz.data,3)
png(paste0(dirTamiFigures,"Cluster52.png"))
fviz_cluster(cz.km,cz.data, ellipse.type = "norm")
dev.off()

## pdf
##   2
```



13.7.2 Particionamiento alrededor de los mediodes

¿Y si no podemos calcular el vector de medias? ¿Y si no tiene sentido calcular el vector de medias? ¿Cómo promediar dos configuraciones de puntos distintas? ¿Cómo promediamos dos formas distintas descritas numéricamente? Cuando el concepto de promedio aritmético no tiene sentido podemos generalizar el procedimiento anterior y hablar de *k-mediodes*.

La idea ahora es sustituir esos centros calculados como vectores de medias de los individuos de un mismo grupo por individuos bien centrados, por individuos típicos, que sustituyan a las medias.

Supongamos que tomamos k individuos de la muestra que denotamos por m_i con $i = 1, \ldots, k$. Particionamos la muestra en k grupos de modo que el grupo C_i está formado por los individuos más próximos a m_i que a cualquier otro m_j con $j \neq i$,

$$C_i = \{l : d(x_l, x_i) = \min_{j \neq i} d(x_l, x_j)\}.$$

Consideremos la siguiente cantidad:

$$\sum_{i=1}^{k} \sum_{j \in C_i} d(x_j, x_{m_i}). \tag{13.5}$$

En el método de particionamiento alrededor de los mediodes nos planteamos encontrar las observaciones $m_1, \ldots, m_k$ que minimizan el valor dado en 13.5.

Ejemplo 13.6 (ruspini) *Aplicamos PAM.*

```
ruspini.pam = pam(ruspini,4)
```

La clasificación obtenida corresponde con la que mostramos en la figura 13.5. Con los datos ruspini cualquiera de los métodos que hemos utilizado cuando decidimos quedarnos con 4 grupos nos da los mismos resultados de clasificacion.

Ejemplo 13.7 (cz.data) *Empezamos con dos grupos.*

```
cz.pam = pam(cz.data,2)
```

Y representamos los resultados de la clasificación en la figura 13.10

```
png(paste0(dirTamiFigures,"Cluster58.png"))
fviz_cluster(cz.pam,cz.data, ellipse.type = "norm")
dev.off()

## pdf
##   2
```

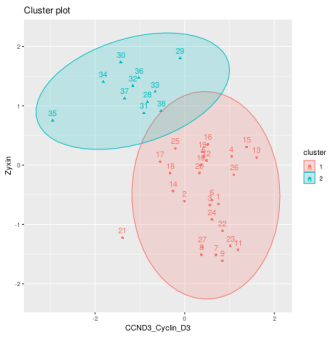


Figura 13.10: Clasificación de los datos cz.data con método PAM y dos grupos.

13.8 Silueta

Veamos cómo se construye la silueta. Para la observación i y el grupo C consideramos

$$\bar{d}(i, C) = \frac{1}{|C|} \sum_{j \in C} d(x_i, x_j),$$

la disimilaridad media i con los elementos del grupo C. Para cada observación i, sea A el cluster al cual lo ha asignado el procedimiento cluster que empleamos y calculamos $a(i)$ la disimilaridad media de i con todos los demás individuos del grupo A, $a(i) = \bar{d}(i, A)$. Obviamente estamos asumiendo que A contiene al menos otro objeto. Consideremos $\bar{d}(i, C)$ para todos los grupos $C \neq A$ y seleccionemos el que tiene el mínimo valor:

$$b(i) = \min_{C \neq A} \bar{d}(i, C).$$

El grupo B donde se alcanza este mínimo, es decir, $\bar{d}(i, B) = b(i)$ se le llama vecino del objeto i. [6] Definimos $s(i)$ como

$$\begin{aligned} s(i) &= 1 - \frac{a(i)}{b(i)} \text{ si } a(i) < b(i), && (13.6)\\ &= 0 \text{ si } a(i) = b(i), && (13.7)\\ &= \frac{b(i)}{a(i)} - 1 \text{ si } a(i) > b(i). && (13.8) \end{aligned}$$

Esto se puede expresar en una única ecuación como

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}.$$

En el caso en que el grupo A contenga un único objeto no está muy claro cómo definir $a(i)$. Tomaremos $s(i) = 0$ que es una elección arbitraria. Se comprueba con facilidad que $-1 \leq s(i) \leq 1$ para cualquier objeto i.

Para interpretar el significado de $s(i)$ es bueno ver los valores extremos. Si $s(i)$ es próximo a uno significa que $a(i)$ es mucho menor que $b(i)$ o lo que es lo mismo, que el objeto i está bien clasificado pues la disimilaridad con los de su propio grupo es mucho menor que la disimilaridad con los del grupo más próximo que no es el suyo. Un valor próximo a cero significa que $a(i)$ y $b(i)$ son similares y no tenemos muy claro si clasificarlo en A o en B. Finalmente un valor de $s(i)$ próximo a -1 significa que $a(i)$ es claramente mayor que $b(i)$. Su disimilaridad media con B es menor que la que tiene con A. Estaría mejor clasificado en B que en A. No está bien clasificado.

Los valores de $s(i)$ aparecerán representados para cada cluster en orden decreciente. Para cada objeto se representa una barra horizontal con longitud proporcional al valor $s(i)$. Una buena separación entre grupos o cluster viene indicada por unos valores positivos grandes de $s(i)$. Además de la representación gráfica se proporciona un análisis descriptivo. En concreto la media de los valores de la silueta dentro de cada cluster y la media de la silueta para todo el conjunto de datos. La clasificación será tanto mejor cuanto mayor sean estos valores medios. De hecho, se puede decidir el número de grupos en función del valor medio de las silueta sobre toda la muestra. Vamos probando distintos números de grupos y nos quedamos con el número que nos da la silueta media máxima.

¿Cuándo podemos decir que hay estructura de grupos en los datos que estamos analizando? Experiencias con datos sugieren la tabla 13.1.

Ejemplo 13.8 (ruspini) *Veamos el resumen de la silueta.*

[6] No parece un nombre inadecuado.

Tabla 13.1: Silueta media y estructura en un conjunto de datos

SC	Interpretación
$0.71 - 1.00$	Fuerte estructura
$0.51 - 0.70$	Estructura razonable
$0.26 - 0.50$	Estructura débil. Probar otros métodos
≤ 0.25	No se encuentra estructura

```
ruspini.pam = pam(ruspini,4)
summary(silhouette(ruspini.pam))

## Silhouette of 75 units in 4 clusters from pam(x = ruspini, k = 4) :
##  Cluster sizes and average silhouette widths:
##        20        23        17        15
## 0.7262347 0.7548344 0.6691154 0.8042285
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.4196  0.7145  0.7642  0.7377  0.7984  0.8549
```

También podemos representarla gráficamente en la figura 13.11.

```
png(paste0(dirTamiFigures,"Cluster63.png"))
fviz_silhouette(ruspini.pam)

##   cluster size ave.sil.width
## 1       1   20          0.73
## 2       2   23          0.75
## 3       3   17          0.67
## 4       4   15          0.80

dev.off()

## pdf
##   2
```

13.9 Análisis cluster y datos de expresión de gen

[194]

En las secciones previas hemos visto algunos métodos de clasificación así como de valoración de hay estructura cluster en un conjunto de datos dados. Estos métodos se aplican (sin modificación) en contextos muy diversos. Sin embargo, ¿qué interés particular tiene su aplicación para datos de expresión de gen? Lo primero a considerar es que nos podemos plantear o bien la clasificación de las filas de la matriz de expresión o bien la clasificación de las columnas. Es decir, tiene sentido biológico clasificar genes o bien las muestras.

Si clasificamos las filas lo que tenemos en cada fila es el perfil de expresión del gen en todas las muestras del estudio. En este caso, puede ser que los grupos que obtengamos nos puedan servir posteriormen-

[194] Muchos comentarios e ideas de esta sección y del resto del capítulo las he tomado de la excelente presentación [40].

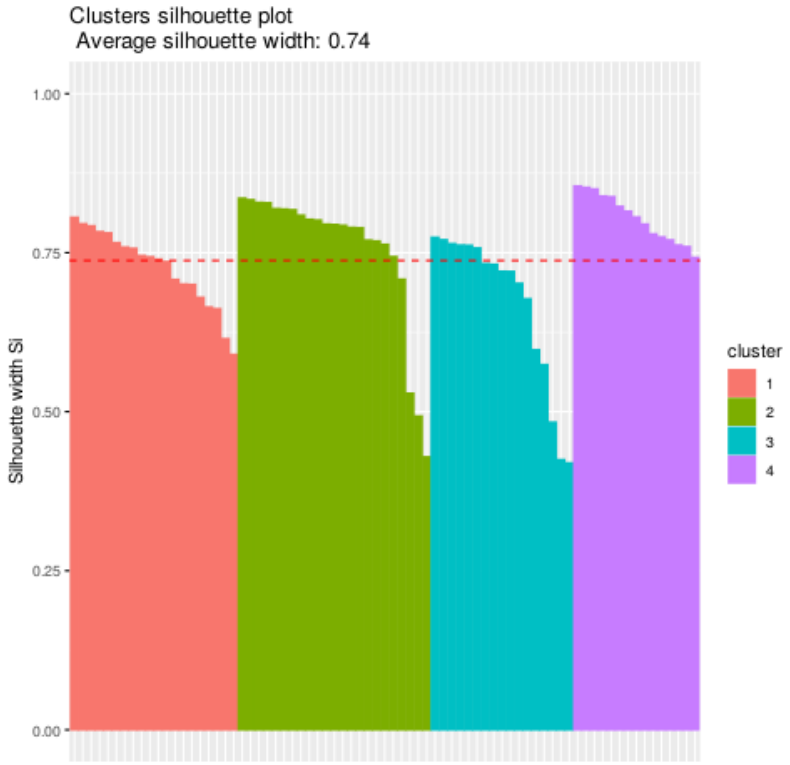


Figura 13.11: Silueta de la clasificación de los datos ruspini con método PAM y cuatro grupos.

te (quizás cruzando información con alguna de las bases de datos de anotación de genes) para realizar un enriquecimiento del conjunto.

Si clasificamos las columnas, las muestras, entonces podemos estar, por ejemplo, buscando subclases de células tumorales. Esto es, tenemos muestras que, a priori, son indistinguibles y posiblemente tengamos alguna subclase que no podíamos preveer.

Los resultados de la clasificación de genes y muestras puede utilizarse como control de calidad. Sobre todo en las muestras tenemos un diseño experimental previo (habitualmente, el diseño grupo control frente a grupo de tratamiento).

13.10 Ejemplos

En esta sección incluimos análisis cluster completos de algunos bancos de datos. Utilizamos herramientas no solamente de este tema sino también de los anteriores.

13.10.1 Un análisis de los datos ALL

Son los datos del paquete [80, ALL] (§ 5.4). Cargamos los datos.

```
data("ALL",package="ALL")
```

Podemos comprobar que es un `ExpressionSet`.

```
class(ALL)

## [1] "ExpressionSet"
## attr(,"package")
## [1] "Biobase"
```

Se realiza el preprocesado de los datos que indicamos en § 5.4.

```
bcell = grep("^B",as.character(ALL$BT))
types = c("NEG","BCR/ABL")
moltyp = which(as.character(ALL$mol.biol) %in% types)
all = ALL[,intersect(bcell,moltyp)]
tipos = ALL$mol.biol[intersect(bcell,moltyp)]
tipos = factor(tipos) ## Eliminamos categorías vacías
```

Vemos que `all` es un `ExpressionSet`.

```
class(all)

## [1] "ExpressionSet"
## attr(,"package")
## [1] "Biobase"
```

Vamos a realizar un filtrado independiente de los genes. En concreto los criterios estadísticos serán que el nivel medio de expresión sea mayor que un cierto valor mínimo tanto en un tipo de biología molecular como en el otro.

```
alltemp0 = genefilter::nsFilter(all,var.func = mean, var.cutoff = 0.7)
alltemp1 = genefilter::nsFilter(alltemp0$eset,var.func = IQR, var.cutoff = 0.7)
all1 = alltemp1$eset
```

```
pacman::p_load(cluster)
all1.d = dist(exprs(all1),method = "manhattan")
all1.km = kmeans(exprs(all1),2)
table(all1.km$cluster)

##
##   1   2
## 272 501

summary(silhouette(all1.km$cluster,all1.d))

## Silhouette of 773 units in 2 clusters from silhouette.default(x = all1.km$cluster, di
##  Cluster sizes and average silhouette widths:
##       272       501
## 0.2847019 0.4033134
## Individual silhouette widths:
##     Min.  1st Qu.   Median     Mean  3rd Qu.
## -0.09864  0.24309  0.41445  0.36158  0.49128
##     Max.
##  0.58111
```

Y mostramos la silueta en figura 13.12.

```
pacman::p_load(factoextra)
sil0 = silhouette(all1.km$cluster,all1.d)
p = fviz_silhouette(sil0)

##   cluster size ave.sil.width
## 1       1  272          0.28
## 2       2  501          0.40
```

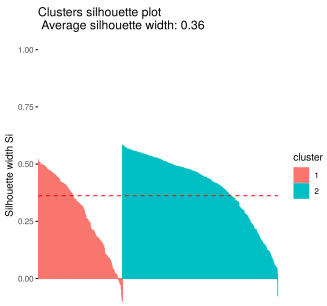


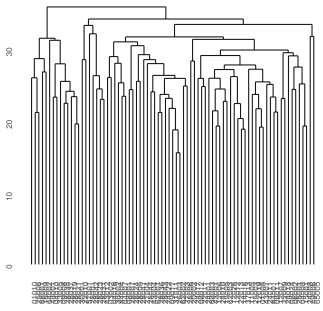
Figura 13.12: Silueta

Realizamos un cluster jerárquico aglomerativo.

```
all1.ag = hclust(all1.d,method="average")
```

O un análisis jerárquico de las muestras.

```
dd = dist(t(exprs(all1)),method="euclidian")## Transpuesta de la matriz
                                            ## de expresión
muestras.ag = hclust(dd,method="average")
p = ggdendro::ggdendrogram(muestras.ag)
```



Recordemos que las muestras correspondían a dos grupos que vienen definidos por la covariable `mol.biol` que nos daba la biología molecular. Veamos si hay coincidencia entre estos tipos y los grupos que hemos obtenido.

```
tipos.ag = cutree(muestras.ag,2)
```

Y construimos la tabla de contingencia.

```
table(tipos,tipos.ag)

##          tipos.ag
## tipos       1  2
##   BCR/ABL 33  4
##   NEG     32 10
```

Mediante un **heatmap** podemos ver conjuntamente los dos dendogramas y una representación gráfica de los niveles de expresión.

```
png(paste0(dirTamiFigures,"Cluster79.png"))
heatmap(exprs(all1),hclustfun=agnes)
dev.off()

## pdf
##   2
```

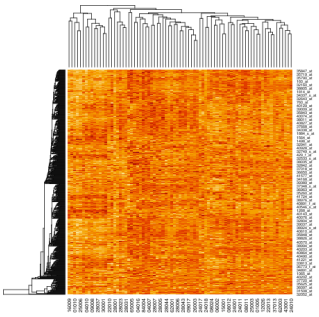


Figura 13.13: Heatmap

Reflexionar qué nos da este dibujo. ¿Para qué nos puede servir un dibujo como este? Básicamente podemos intentar ver si hay *grupos formados por filas (genes) y columnas (muestras).*

Supongamos que nos planteamos un PAM y un k-medias con dos grupos para las muestras. Vamos a clasificar con cada uno de los métodos e intentar ver si hay relación con la variabel **tipos** que contiene la biología molecular.

```
all1.pam = pam(t(exprs(all1)),2)
table(tipos,all1.pam$cluster)

##
## tipos       1  2
##   BCR/ABL 26 11
##   NEG     24 18
```

```
all1.km = kmeans(t(exprs(all1)),2)
table(tipos,all1.km$cluster)

##
## tipos       1  2
##   BCR/ABL 10 27
##   NEG     21 21
```

No parece que tengamos ninguna relación clara. Como siempre hay que seguir trabajando los datos a ver si se encuentra algo.

13.10.2 Análisis cluster de GSE20986

Utilizamos los datos preprocesados. Los cargamos.

```
data(gse20986,package="tamidata")
gse = gse20986
```

```
library(genefilter)
gse0 = nsFilter(gse,var.func = mean, var.cutoff = 0.7)
gse1 = nsFilter(gse0$eset,var.func = IQR, var.cutoff = 0.7)
gse = gse1$eset
```

¿Con cuántas filas nos hemos quedado?

```
dim(exprs(gse))

## [1] 1816   12
```

Vamos a clasificar las muestras y compararemos con la clasificación original de las mismas. Utilizamos distancia Manhattan y promedio con el método PAM.

Primero calculamos la matriz de distancias.

```
d0 = dist(t(exprs(gse)),method = "manhattan")
```

Y aplicamos PAM.

```
library(cluster)
gse.pam = pam(d0,diss = TRUE,k=4)
```

La clasificación original es

```
tipo0 = rep(1:4,rep(3,4))
```

```
table(tipo0,gse.pam$clustering)

##
## tipo0 1 2 3 4
##     1 1 1 1 0
##     2 2 1 0 0
##     3 3 0 0 0
##     4 0 0 0 3
```

Lo que sale es muy interesante y tiene una interpretación.

13.11 Otras aproximaciones y problemas

Sin duda, un problema de gran interés es lo que se conoce como biclustering o co-clustering. En este caso intentamos grupos de filas (genes) y columnas (muestras) que muestren unos valores de expresión similares. Es un problema importante pero no lo tratamos (de momento) en estas notas. Una referencia fundamental de biclustering es [88].

La clasificación es tanto mejor cuanta más información se utiliza. En particular un trabajo de interés para clasificación de genes incorporando información de Gene Ontology es [137].

Cuando se clasifican muestras el mayor problema es la enorme dimensión de los vectores que clasificamos (N en nuestro caso). Una posibilidad es seleccionar genes y no utilizarla la expresión para todos ellos. En [2] se propone un método utilizando información de Gene Ontology.

13.12 Ejercicios

Ejercicio 26

Utilizando los datos *ALL* se pide:

1. Filtrar las muestras (columnas) con biología molecular "BCR/ABL" y "NEG".
2. Quedarse con los genes (filas) que verifiquen las siguientes condiciones:
   (a) La media de los niveles de expresión sobre las biologías moleculares indicadas sea superior a 5.62.
   (b) La desviación estándar de las expresiones del gen ha de ser superior a 1.85.
3. Aplicar un análisis cluster jerárquico a los genes. En concreto:
   (a) Hay que representar el dendograma.
   (b) Decidir por inspección visual qué número de grupos parece razonable. Supongamos que denotamos por k el número de grupos.
   (c) Representar la silueta y obtener su anchura media.
4. Aplicar un análisis cluster jerárquico a las muestras. En concreto:
   (a) Hay que representar el dendograma.
   (b) Decidir por inspección visual qué número de grupos parece razonable. Supongamos que denotamos por k el número de grupos.
   (c) Representar la silueta y obtener su anchura media.
5. En este apartado vamos a aplicar un k-medias. En concreto vamos a ir clasificando según este método y nos quedaremos con el número de grupos que tengan una anchura media de la silueta mayor.
6. Repetir el apartado anterior utilizando el método PAM. Discutir el número de grupos.
7. Comparar las clasificaciones obtenidas en los dos apartados anteriores.

Parte V

Análisis de grupos de genes

Capítulo 14

Grupos de genes

14.1 Introducción

Hasta ahora hemos trabajado habitualmente a nivel de gen. Cada gen en cada muestra nos da un nivel de expresión. Obviamente esto supone que por muestra tenemos una muy alta dimensión. Un análisis de expresión diferencial supone que tenemos un contraste por gen. Muchos miles de contrastes supone esto. Si pretendemos clasificar la muestra en distintos fenotipos entonces la dimensión alta con la que trabajamos produce un sobreajuste y modelos que no son generalizables a otros datos. En ambos problemas no es bueno tener una dimensión tan alta. Podemos utilizar componentes principales para resumir el perfil de expresión de la muestra (§ 12). Otra opción puede ser trabajar no con genes individuales sino con *g*rupos de genes. Si nos interesa la expresión diferencial entre condiciones entonces podemos valorar si el grupo de genes que nos interese, como grupo, se expresa diferencialmente. Si vamos a clasificar en fenotipos, podemos resumir la expresión de los genes en una muestra dentro de cada grupo. Reducimos el número de hipótesis a contrastar (expresión diferencial) o bien la dimensión del vector que utilizamos para clasificar (clasificación en fenotipo).

Obviamente podemos construir el grupo de genes que nos apetezca y evaluarlo. No parece algo muy lógico. Es de sobra conocido en la literatura estadística que un contraste de hipótesis es útil en la misma medida en que está correctamente formulado. Y formular correctamente el contraste supone conocimiento sobre el problema que abordamos. En este caso la comunidad científica ha propuesto y propone y propondrá clasificaciones que se han de utilizar para definir estos grupos.

Tres fuentes vamos a utilizar a la hora de considerar grupos de genes.

Gene Ontology Podemos utilizar las tres ontologías consideradas en Gene Ontology. El grupo de genes viene definido por todos aquellos genes que tienen este término.

KEGG Tenemos las rutas de señalización para muchos organismos.

MSigDB En MSigDB [195] consideran hasta ocho formas distintas de construir estos grupos.

[195] The Molecular Signatures Database

En http://www.genesetdb.auckland.ac.nz/haeremai.html tenemos una herramienta web que contiene a su vez a otras muchas bases de datos. Realiza análisis de sobre representación y está orientada a humanos, ratones y ratas.

Como vemos hay muchas formas de definir estos grupos. En cualquier caso en su definición interviene un conocimiento previo. No utilizamos los propios datos de expresión con los que posteriormente vamos a trabajar.[196]

[196] No son grupos obtenidos por un análisis cluster previo.

En este tema tratamos de cómo construir grupos de genes utilizando R/Bioconductor. Es un tema de carácter muy técnico previo al que sigue en donde analizaremos la posible expresión diferencial de estos grupos.[197]

[197] Siempre nuestro interés principal.

Vamos a definir conjuntos de genes o colecciones de conjuntos de genes utilizando el paquete [94, GSEABase].

```
library(GSEABase)
```

14.2 Homo sapiens

La opción más simple sería definirnos nuestro propio conjunto de genes. Para ello podemos utilizar la función GSEAbase::GeneSet. Tomamos como ejemplo GSE20986 (§ 5.11).

```
data(gse20986,package="tamidata")
eset = gse20986
```

Supongamos que queremos construir un conjunto de genes formado por aquellos que ocupan las filas de la 345 a la 405 (sin ningún sentido práctico). Los índices los podemos obtener con

```
345:405
```

```
##  [1] 345 346 347 348 349 350 351 352 353 354 355
## [12] 356 357 358 359 360 361 362 363 364 365 366
## [23] 367 368 369 370 371 372 373 374 375 376 377
## [34] 378 379 380 381 382 383 384 385 386 387 388
## [45] 389 390 391 392 393 394 395 396 397 398 399
## [56] 400 401 402 403 404 405
```

Construimos el grupo y le damos nombre.

```
(egs = GeneSet(eset[345:405, ], setName = "Burjasot"))
```

Los genes tienen sus identificadores obtenidos de su anotación (mostramos los primeros).

```
head(geneIds(egs))
```

```
## [1] "1552739_s_at" "1552740_at"   "1552742_at"
## [4] "1552743_at"   "1552745_at"   "1552747_a_at"
```

En este caso, como se obtuvo con Affymetrix GeneChip, los identificadores son los proporcionados por el fabricante y corresponde a los nombres de los conjuntos de sondas.[198] Podemos tener más información con

[198] Ver § 4.

```
details(egs)

## setName: Burjasot
## geneIds: 1552739_s_at, 1552740_at, ..., 1552822_at (total: 61)
## geneIdType: Annotation (hgu133plus2)
## collectionType: ExpressionSet
## setIdentifier: debian:5900:Thu May  7 20:25:47 2020:26335
## description:
## organism: Homo sapiens
## pubMedIds:
## urls:
## contributor:
## setVersion: 0.0.1
## creationDate:
```

En el ejemplo que acabamos de ver hemos creado un conjunto de genes a partir de un ExpresionSet pero podemos hacerlo de otros modos.

```
showMethods("GeneSet", inherited = FALSE)

## Function: GeneSet (package GSEABase)
## type="BroadCollection"
## type="character"
## type="ExpressionSet"
## type="GeneIdentifierType"
## type="GOCollection"
## type="missing"
```

Podemos obtener la correspondencia de nuestros identificadores con otros tipos de identificadores, por ejemplo, los EntrezId:

```
mapIdentifiers(egs, EntrezIdentifier(), verbose = TRUE)
```

También podemos tener una colección de conjuntos, GSEABase::GeneSetCollection. Quizás la mejor manera de definir conjuntos utilizando un ExpresionSet con su anotación es hacerlo utilizando Gene Ontology u otra base de datos. Por ejemplo, en el siguiente ejemplo construimos los conjuntos para nuestro ExpressionSet `eset`.

```
gsc = GeneSetCollection(eset, setType = GOCollection())
```

Podemos ver un resumen de los conjuntos.

```
gsc

## GeneSetCollection
##   names: GO:0000002, GO:0000011, ..., GO:2001070 (17312 total)
##   unique identifiers: 1555591_at, 201917_s_at, ..., 1552675_at (37100 total)
##   types in collection:
##     geneIdType: AnnotationIdentifier (1 total)
##     collectionType: GOCollection (1 total)
```

Por ejemplo, el correspondiente a `GO:0000122` sería

```
gsc[["GO:0000122"]]

## setName: GO:0000122
## geneIds: 1487_at, 1552338_at, ..., AFFX-HUMISGF3A/M97935_MB_at (total: 1936)
## geneIdType: Annotation (hgu133plus2)
## collectionType: GO
##   ids: GO:0000122 (1 total)
##   evidenceCode: EXP IDA IPI IMP IGI IEP ISS ISO ISA ISM IGC IBA IBD IKR IRD RCA TAS
##   ontology: CC MF BP
## details: use 'details(object)'
```

Ahora vamos convertir estos mismos conjuntos a identificadores ENTREZ.

```
gsc = mapIdentifiers(gsc, EntrezIdentifier())
```

Podemos ver el primero de ellos.

```
head(geneIds(gsc),n=1)

## $`GO:0000002`
##  [1] "80119" "55186" "291"   "4358"  "1890"
##  [6] "4205"  "9361"  "4976"  "10000" "63875"
## [11] "84275" "92667"
```

¿Cuántos elementos tiene cada uno de los conjuntos que hemos construido?

```
head(sapply(geneIds(gsc), length))
```

Podemos quedarnos con los dos grupos que tengan un cardinal mínimo, por ejemplo, por encima de 10.

```
gsc.filt = gsc[sapply(geneIds(gsc), length) > 10]
```

Son los siguientes grupos.

```
geneIds(gsc.filt)
```

14.3 Grupos con levadura

Veamos cómo construir grupos de genes utilizando los términos GO en la Saccharomyces cerevisiae. Empezamos cargando el fichero de anotación.

```
library(org.Sc.sgd.db)
```

```
frame = toTable(org.Sc.sgdGO)
goframeData = data.frame(frame$go_id, frame$Evidence, frame$systematic_name)
goFrame = GOFrame(goframeData, organism = "Saccharomyces cerevisiae")
goAllFrame = GOAllFrame(goFrame)
gscSc = GeneSetCollection(goAllFrame, setType = GOCollection())
```

Y lo guardamos.

Habitualmente cuando queramos utilizar estos grupos tendremos que hacer dos cosas:

1. Quedarnos con aquellos genes que aparecen en nuestra plataforma.
2. Quedarnos posiblemente con grupos de genes con un tamaño mínimo.

Para ello vamos a utilizar la siguiente función.

```
subsettingGeneSet = function(gs0, fn0) {
    geneIds(gs0) = geneIds(gs0)[is.element(geneIds(gs0), fn0)]
    gs0
}
```

Y ahora podemos modificar el GeneSetCollection de modo que nos quedamos con los genes que tenemos en nuestros datos tamidata::gse6647.

```
data(gse6647,package="tamidata")
```

```
gsc1 = sapply(gscSc, subsettingGeneSet, fn0 = featureNames(gse6647))
gsc2 =  GeneSetCollection(gsc1)
```

14.4 Grupos para GSE1397

Leemos los datos.

```
data(gse1397,package="tamidata")
```

Cargamos la anotación del ExpressionSet para poder formar grupos.

```
library(annotate)
annotation(gse1397)

## [1] "hgu133a"

library(hgu133a.db)
```

Buscamos los conjuntos de genes utilizando Gene Ontology.[1]

```
library(GSEABase)
```

```
gse1397.gsc = GeneSetCollection(gse1397,setType=GOCollection())
names(gse1397.gsc) = unlist(lapply(gse1397.gsc,setName))
```

¿Cuántos grupos tenemos definidos?

[1] Se lleva un tiempo así que paciencia.

```
gsc = gse1397.gsc
length(gsc)

## [1] 16445
```

Como vemos muchos. Sin embargo, la mayor parte de ellos tiene muy pocos genes. Podemos ver información del primer grupo.

```
(g1 = gsc[[1]])

## setName: GO:0000002
## geneIds: 201917_s_at, 201918_at, ..., 222216_s_at (total: 18)
## geneIdType: Annotation (hgu133a)
## collectionType: GO
##   ids: GO:0000002 (1 total)
##   evidenceCode: EXP IDA IPI IMP IGI IEP ISS ISO ISA ISM IGC IBA IBD IKR IRD RCA TAS
##   ontology: CC MF BP
## details: use 'details(object)'
```

Y sus identificadores (AffyId) son

```
geneIds(g1)

##  [1] "201917_s_at" "201918_at"   "201919_at"
##  [4] "202825_at"   "203466_at"   "204858_s_at"
##  [7] "208328_s_at" "209017_s_at" "212213_x_at"
## [10] "212214_at"   "212607_at"   "212609_s_at"
## [13] "214306_at"   "214684_at"   "214821_at"
## [16] "217497_at"   "219393_s_at" "222216_s_at"
```

Sus identificadores entrez o Ensembl son

```
unlist(mget(geneIds(g1),hgu133aENTREZID))

## 201917_s_at   201918_at   201919_at   202825_at
##     "55186"     "55186"     "55186"       "291"
##   203466_at 204858_s_at 208328_s_at 209017_s_at
##      "4358"      "1890"      "4205"      "9361"
## 212213_x_at   212214_at   212607_at 212609_s_at
##      "4976"      "4976"     "10000"     "10000"
##   214306_at   214684_at   214821_at   217497_at
##      "4976"      "4205"       "291"      "1890"
## 219393_s_at 222216_s_at
##     "10000"     "63875"

unlist(mget(geneIds(g1),hgu133aENSEMBL))

##       201917_s_at         201918_at
## "ENSG00000114120" "ENSG00000114120"
##         201919_at         202825_at
## "ENSG00000114120" "ENSG00000151729"
##         203466_at       204858_s_at
## "ENSG00000115204" "ENSG00000025708"
##       208328_s_at       209017_s_at
## "ENSG00000068305" "ENSG00000196365"
```

```
##       212213_x_at         212214_at
## "ENSG00000198836" "ENSG00000198836"
##        212607_at1        212607_at2
## "ENSG00000117020" "ENSG00000275199"
##      212609_s_at1      212609_s_at2
## "ENSG00000117020" "ENSG00000275199"
##         214306_at         214684_at
## "ENSG00000198836" "ENSG00000068305"
##         214821_at         217497_at
## "ENSG00000151729" "ENSG00000025708"
##      219393_s_at1      219393_s_at2
## "ENSG00000117020" "ENSG00000275199"
##       222216_s_at
## "ENSG00000158042"
```

La mayor parte de estos grupos son muy pequeños. Lo siguiente nos da el tamaño de estos grupos.

```
head(table(sapply(geneIds(gsc),length)))

##
##    1    2    3    4    5    6
## 2999 2047 1601 1263  983  711
```

Podemos ver que de tamaño uno tenemos 2336 grupos. Nos quedamos con aquellos grupos que, al menos, tienen 50 genes.

```
gse1397.gsc.filt = gsc[which(sapply(geneIds(gsc),length) > 50)]
```

14.5 Grupos GO para levadura

```
library(org.Sc.sgd.db)
frame = toTable(org.Sc.sgdGO)
goframeData = data.frame(frame$go_id, frame$Evidence, frame$systematic_name)
goFrame = GOFrame(goframeData, organism = "Saccharomyces cerevisiae")
goAllFrame = GOAllFrame(goFrame)
gscSc = GeneSetCollection(goAllFrame, setType = GOCollection())
save(gscSc,file=paste0(dirTamiData,"gscSc.rda"))
```

En lo que sigue será frecuente que no queramos utilizar todos los grupos que hemos construido. ¿Cómo quedarnos con aquellos que tienen al menos un número dado de genes? Por ejemplo, quedarnos con los grupos con 10 o más genes. El siguiente código lo hace.

```
gruposGrandes = which(sapply(geneIds(gscSc),length) > 10)
gsc1 = gscSc[gruposGrandes]
```

Podemos ver el primer grupo.

```
gsc1[[1]]
```

```
## setName: GO:0000001
## geneIds: YNL304W, YHR194W, ..., YNR035C (total: 29)
## geneIdType:
## collectionType: GO
##   ids: GO:0000001 (1 total)
##   evidenceCode: EXP IDA IPI IMP IGI IEP ISS ISO ISA ISM IGC IBA IBD IKR IRD RCA TAS
##   ontology: CC MF BP
## details: use 'details(object)'
```

14.6 Grupos GO para humanos

En esta sección construimos los grupos de genes según Gene Ontology para humanos. El código es el siguiente:[199]

[199] Para otros organismos sería similar reemplazando el paquete [28] por el correspondiente al organismo que nos interese.

```
library("org.Hs.eg.db")
frame = toTable(org.Hs.egGO)
goframeData = data.frame(frame$go_id, frame$Evidence, frame$gene_id)
goFrame = GOFrame(goframeData, organism = "Homo sapiens")
goAllFrame = GOAllFrame(goFrame)
gscHs = GeneSetCollection(goAllFrame, setType = GOCollection())
save(gscHs,file = paste0(dirTamiData,"gscHs.rda"))
```

14.7 Grupos para Arabidopsis thaliana

Utilizamos el paquete [24].

```
pacman::p_load("ath1121501.db")
frame = toTable(org.At.tairGO)
goframeData = data.frame(frame$go_id, frame$Evidence, frame$gene_id)
goFrame = GOFrame(goframeData, organism = "Arabidopsis")
goAllFrame = GOAllFrame(goFrame)
gscAt = GeneSetCollection(goAllFrame, setType = GOCollection())
gscAt =  geneIds(gscAt)
save(gscAt,file = paste0(dirTamiData,"gscAt.rda"))
```

14.8 Ejercicios

Ej. 27 — Construir los grupos basados en Gene Ontology y en KEGG para el ratón (Mus musculus).

Ej. 28 — Construir los grupos basados en Gene Ontology y en KEGG para las cepas de la E. coli que puedas.

Ej. 29 — Consultar las anotaciones OrgDb para decidir de qué organismos podemos construir los grupos.

Capítulo 15

Test de Fisher unilateral

15.1 Introducción

Por alguna razón (que desconozco) a este procedimiento se le llama en numerosas publicaciones de perfil biológico *test hipergeométrico*. [1] En lo tratado hasta este momento hemos obtenido una ordenación de los genes. Esta lista la hemos estudiado pretendiendo que cuando mayor sea la expresión diferencial del gen este aparezca antes en la lista. De este modo el primer gen es el *marginalmente* (o si se prefiere individualmente) tiene una mayor expresión diferencial y así sucesivamente. De hecho, el p-valor no es más que una medida de esa diferenciación. Una expresión muy utilizada en la literatura es que que hay una asociación entre el gen (su expresión) y el fenotipo.[200] De alguna forma el p-valor que obtenemos en cada test es una cuantificación de la asociación gen-fenotipo. Luego modificamos estos p-valores de forma que se tiene en cuenta todos los genes que simultáneamente se están estudiando. Obtenemos de este modo unos p-valores ajustados. Finalmente, tanto los p-valores originales como los ajustados no dejan de ser cuantificaciones marginales de la asociación gen-fenotipo. Cuando hemos fijado una tasa de error (FDR o FWER) lo que hacemos es fijar un punto de corte. En esa lista que hemos construido decidimos (con algún criterio de error) en qué punto de la lista cortamos. Los genes que están antes del punto de corte se consideran significativos y los que siguen no. Reducimos nuestra información a un si (gen significativo o que tiene expresión diferencial o que hay asociación gen-fenotipo[2] o un no (no es significativo, no hay expresión diferencial o no hay asociación gen-fenotipo).

[200] La expresión fenotipo se usa de un modo muy amplio porque podemos estar hablando de características fenotípicas o simplemente un diseño experimental con factores temporales o diferentes temperaturas.

Ya tenemos ese conjunto de genes significativos. Incluso hemos visto cómo generar unos enlaces para que gen a gen examinemos alguna base de datos online. Es claro que el investigador puede ir generando hipótesis sobre qué indica esta lista. Pero claramente no es una labor simple. Una posible ayuda puede ser utilizar información previamente generada por la comunidad científica sobre grupos de genes. Grupos que indican que tienen relación con una misma función, o que están localizados próximos en un mismo cromosoma. En fin, grupos con

[1] La razón de este nombre sí la entiendo (la distribución del estadístico de contraste es la distribución hipergeométrica) lo que no entiendo es el cambio de nombre ya que la denominación test de Fisher está más que consolidada y debiera de utilizarse.

[2] A gusto del consumidor.

Tabla 15.1: S_0 indica el grupo de genes significativos (grupo predefinido) y $G \setminus S_0$ su complementario respecto del universo de genes considerado G. S_1 indica el conjunto de genes contra el cual comparamos.

	S_1	$S_1^c = G \setminus S_1$	
S_0	n_{11}	n_{12}	$n_{1\cdot}$
$S_0^c = G \setminus S_0$	n_{21}	n_{22}	$n_{2\cdot}$
	$n_{\cdot 1}$	$n_{\cdot 2}$	N

sentido (biológico).

Un poco de notación que nunca es mala. Sea G es el conjunto de genes considerado. ¿Quién es este conjunto? En un estudio con microarrays uno diría que es el conjunto de genes que se está explorando. Sin embargo, un chip suele estar diseñado para observar tanto genes como se pueda y no hay una selección previa. Quizás no sea muy razonable considerar el conjunto total de genes este. En otros casos no es así. Lo fundamental es darse cuenta que lo que hacemos depende de un modo esencial del conjunto total de genes considerado o universo. S_0 ($\subset G$) el conjunto de genes que *nuestro* estudio ha indicado como significativo y S_1 un conjunto de genes predefinido (misma función, misma localización). Un primer problema es definir los conjuntos S_1 con los que comparar.[201] Una vez definidos *nuestro conjunto* (S_0), el conjunto con el que queremos comparar (S_1) y el conjunto total de genes considerado (G) la situación que se presenta la tenemos reflejada en la tabla 15.1. En esta tabla n_{11} indica el número de genes que están tanto en S_0 como en S_1, también tendremos n_{12} genes que están en S_0 pero no en S_1, n_{21} en S_1 pero no en S_0 y, finalmente, n_{22} que no están ni en S_0 ni en S_1. Suponemos que el total de genes (nuestro universo de genes) G tiene un total de N genes.

[201] § 14.

En la tabla 15.1 consideramos dados los totales de la fila y la columna, en otras palabras, consideramos fijos los valores de $n_{1\cdot}$ y $n_{2\cdot}$ (totales de fila) así como los valores de $n_{\cdot 1}$ y $n_{\cdot 2}$ (totales de columna). Asumiendo fijos estos totales de fila y columna: ¿cuál es la probabilidad de observar la tabla 15.1? Utilizando argumentos combinatorios la respuesta es la siguiente: Si denotamos N_{11} el número *aleatorio* de genes en común entonces, bajo la hipótesis de que no hay ningún tipo de asociación entre fila y columna, entonces la probabilidad sería

$$P(N_{11} = n_{11}) = \frac{\binom{n_{1\cdot}}{n_{11}}\binom{n_{2\cdot}}{n_{\cdot 1} - n_{11}}}{\binom{N}{n_{\cdot 1}}}.$$

Supongamos que estamos contrastando la posible sobrerepresentación entonces, bajo la hipótesis de independencia (condicionada a las marginales), rechazaríamos la hipótesis de independencia para un valor mayor o igual al observado por lo que el p-valor sería la suma de las probabilidades siguientes

$$p = P(N_{11} \geq n_{11}) = \sum_{t=n_{11}}^{\min\{n_{1\cdot}, n_{\cdot 1}\}} \frac{\binom{n_{1\cdot}}{t}\binom{n_{2\cdot}}{n_{\cdot 1} - t}}{\binom{N}{n_{\cdot 1}}}.$$

Tabla 15.2: S_0 (S) indica el grupo de genes significativos (grupo predefinido) y S_0 (S^c) su complementario. S_1 indica el conjunto de genes contra el cual comparamos.

	S_1	S_1^c	
S_0	30	40	70
S_0^c	120	156	276
	150	196	346

15.2 fisher.test

Supongamos que hemos observado la tabla 15.2 y pretendemos saber si el solapamiento entre ambos conjuntos de genes es mayor que el esperable por el puro azar aunque no estén asociados ambos conjuntos.

Podemos utilizar el test exacto de Fisher direccional (o unilateral o de una cola)

```
conteos = matrix(c(30,120,40,156),ncol=2)
fisher.test(conteos,alternative = "greater")

##
##  Fisher's Exact Test for Count Data
##
## data:  conteos
## p-value = 0.589
## alternative hypothesis: true odds ratio is greater than 1
## 95 percent confidence interval:
##  0.6018802       Inf
## sample estimates:
## odds ratio
##  0.9750673
```

¿Y si valoramos una baja representación?

```
fisher.test(conteos,alternative = "less")

##
##  Fisher's Exact Test for Count Data
##
## data:  conteos
## p-value = 0.5179
## alternative hypothesis: true odds ratio is less than 1
## 95 percent confidence interval:
##  0.000000 1.571751
## sample estimates:
## odds ratio
##  0.9750673
```

Observamos que en ambas salidas nos muestra el cociente de los odds. Un valor del cociente de odds mayor que la unidad indica sobre expresión mientras que un valor menor a la unidad indica una expresión menor a la esperable bajo independencia.

Tabla 15.3: S_0 (respectivamente S_1) indica el grupo de genes significativos (el grupo predefinido) y $G \setminus S_0$ ($G \setminus S_1$) su complementario. El universo de genes crece y suponemos que añadimos k genes.

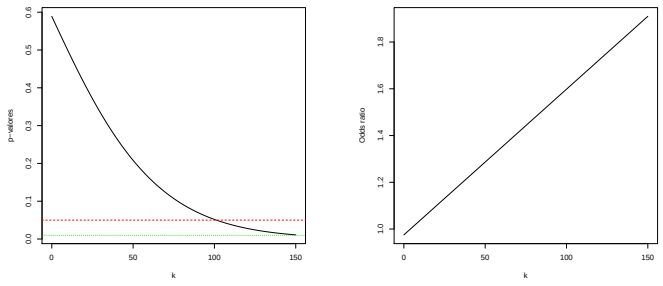
	S_1	$G \setminus S_1$	
S_0	30	40	70
$G \setminus S_0$	120	156 + k	276 + k
	150	196 + k	346 + k

15.3 Sobre la elección del universo de genes

Pretendemos evaluar el efecto que tiene la elección del universo de genes. ¿Qué efecto tiene en nuestro análisis el universo de genes, el conjunto G total de genes que estamos utilizando? Consideremos los datos de la tabla 15.3. ¿Qué describe la tabla? Supongamos que tenemos dos conjuntos de genes dados, S_0 y S_1. Dado este par de grupos de genes tenemos el número de genes en los dos y en uno de ellos pero no en el otro. Si miramos la tabla 15.3 significa que tenemos perfectamente definidas tres entradas de la tabla. Pero si suponemos que vamos incrementando nuestro universo de genes (sin incluir ningún gen adicional en S_0 o S_1) entonces la entrada n_{22} en la tabla 15.3 será cada vez mayor. En concreto hemos supuesto que $n_{22} = 156 + k$, es decir, el conteo original más los k genes que vamos incorporando al universo de genes. Vamos a tomar un valor de k creciente y representaremos el p-valor del test de Fisher unilateral. Vemos cómo conforme el valor de k crece el p-valor decrece (figura 15.1(a)) y el cociente de odds crece (figura 15.1(b)). En definitiva la interpretación tanto del p-valor como del cociente de odds depende del universo de genes que estamos considerando.

En la figura 15.1(a) (respectivamente en la figura 15.1(b)) tenemos el p-valor del test de Fisher unilateral correspondiente a la tabla 15.3 (respectivamente el cociente de odds) cuando el valor de k va 0 a 100. En trazo rojo discontinuo representamos la línea horizontal para un valor de ordenada igual a 0.05. La línea verde punteada corresponde con una ordenada de 0.01.

Vemos cómo *el incremento del universo de genes tiene como consecuencia que el p-valor del test de Fisher unilateral decrezca.*



(a) (b)

Figura 15.1: a) p-valores del test de Fisher unilateral. b) Cocientes de odds.

15.4 Utilizando Category y GOstats

En esta sección utilizamos los paquetes [56, GOstats] (ver también [44]) y [55, Category].

15.4.1 GSE1397

Leemos los datos normalizados.

```
pacman::p_load(Biobase)
data(gse1397,package = "tamidata")
eset = gse1397
y = pData(eset)[,"type"]
remove(gse1397) ## Ahorramos memoria
```

Determinamos grupo de genes significativos.

```
pacman::p_load(genefilter,multtest)
tt = rowttests(eset,y)
p0 = tt$p.value
p1 = mt.rawp2adjp(p0, "BH")
orden.original = order(p1$index)
p.BH = p1$adjp[orden.original,2]
significativos  = which(p.BH < 0.05)
```

¿Qué anotación tienen?[202]

[202] Es una base de datos asociada a un chip.

```
annotation(eset)

## [1] "hgu133a"
```

Cargamos el paquete con la base de datos correspondiente.

```
library(hgu133a.db)
```

¿Tenemos un solo identicador GO para cada gen?

```
G1.entreizd =  unlist(mget(featureNames(eset), hgu133aENTREZID))
anyDuplicated(G1.entreizd)

## [1] 6
```

Hay duplicados. Nos quedamos, para cada gen, con el conjunto de sondas que nos da la máxima variabilidad, por ejemplo, el valor más grande del rango intercuartílico.

```
eset.iqr = apply(exprs(eset), 1, IQR)
uniqGenes = findLargest(featureNames(eset), eset.iqr, "hgu133a")
eset1 = eset[uniqGenes, ]
```

Y ahora construimos el universo de genes y comprobamos que no hay duplicidades.

```
G2.entrezid = unlist(mget(featureNames(eset1), hgu133aENTREZID))
anyDuplicated(G2.entrezid)

## [1] 0
```

Determinamos la identificación en la misma base de datos de los genes declarados significativos.

```
seleccionados = unlist(mget(featureNames(eset[significativos,]),
    hgu133aENTREZID))
```

Vamos a aplicar un test de Fisher unilateral para los grupos definidos de acuerdo con Gene Ontology. Cargamos paquetes necesarios.

```
pacman::p_load(GO.db,Category,GOstats)
```

Y realizamos los tests.[203]

[203] En http://geneontology.org/ tenemos una aplicación en línea que nos realiza un análisis similar.

```
params = new("GOHyperGParams", geneIds = seleccionados,
             universeGeneIds = G2.entrezid,
             annotation = annotation(eset), ontology = "BP",
             pvalueCutoff = 0.001,conditional = FALSE,
             testDirection = "over")
overRepresented = hyperGTest(params)
```

Finalmente para visualizar los resultados guardamos los resultados en un fichero y lo vemos con el navegador.

```
htmlReport(overRepresented, file = "GSE1397overRepresented.html")
browseURL("GSE1397overRepresented.html")
```

También podemos ver un resumen.

```
head(summary(overRepresented))
```

Con el siguiente código obtenemos un grafo donde los vértices corresponden a los grupos definidos por la categorías Gene Ontology significativas. Las aristas nos muestran la estructura de la base de datos que es un grafo acíclico dirigido.

```
library(Rgraphviz)
plot(goDag(overRepresented))
```

```
library(Rgraphviz)
png(file=paste(dirTamiFigures,"GSE1397overRepresented.png",sep=""))
plot(goDag(overRepresented))
dev.off()
```

15.4.2 GSE20986

Leemos los datos normalizados y lo renombramos.

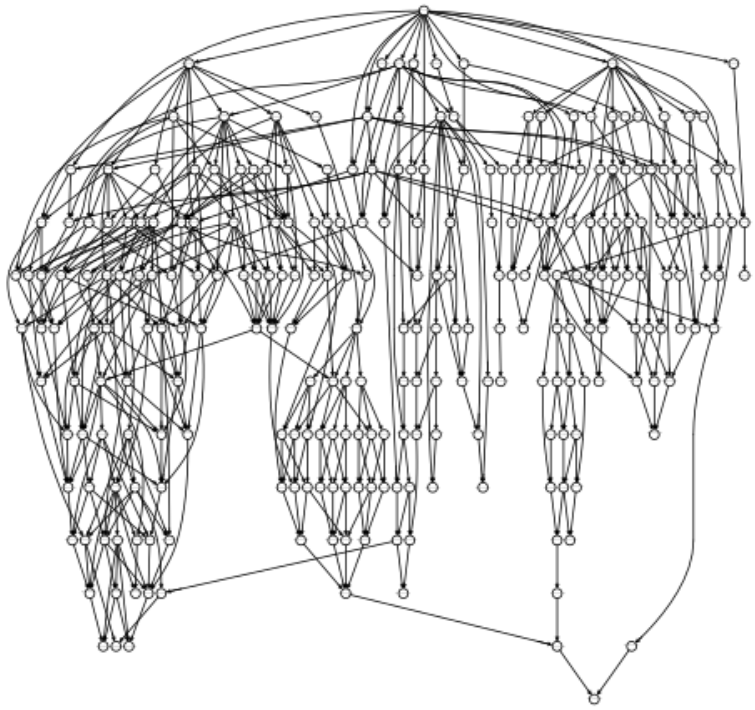


Figura 15.2: Grafo Gene Ontology con las grupos significativos.

```
data(gse20986, package = "tamidata")
gse = gse20986
```

En lo que sigue necesitamos la anotación de este ExpressionSet.

```
Biobase::annotation(gse)

## [1] "hgu133plus2"
```

Cargamos el paquete con la base de datos correspondiente.

```
library("hgu133plus2.db")
```

Lo primero: **definir el universo de genes**. ¿Cómo? Una propuesta razonable[3] podría ser aplicar un filtrado no específico y que el universo de genes sean los restantes.

```
library(genefilter)
gse.filt = genefilter::nsFilter(gse, var.func = IQR, var.cutoff = 0.6,
                    require.GOBP = TRUE)$eset
```

¿Cuántos genes nos quedan?

[3] Y nada más que esto razonable y tan razonable como muchas otras.

```
dim(gse.filt)

## Features  Samples
##     6209       12
```

¿Tenemos un solo identicador GO para cada gen?

```
G1.entreizd = unlist(mget(featureNames(gse.filt), hgu133plus2ENTREZID))
anyDuplicated(G1.entreizd)

## [1] 0
```

Vemos que no hay duplicidades en los datos filtrados. Sin embargo, si consideramos como universo de genes todos los del chip entonces sí que nos encontramos con duplicidades.

```
G2.entrezid = unlist(mget(featureNames(gse), hgu133plus2ENTREZID))
anyDuplicated(G2.entrezid)

## [1] 6
```

La idea sería quedarnos para cada gen con el conjunto de sondas que nos da la máxima variabilidad. Por ejemplo, con el grupo de sondas con el mayor rango intercuartílico.

```
gse.iqr = apply(exprs(gse), 1, IQR)
uniqGenes = findLargest(featureNames(gse), gse.iqr, "hgu133plus2")
gse.filt2 = gse[uniqGenes, ]
```

Y ahora construimos el universo de genes y comprobamos que no hay duplicidades.

```
G2.entrezid = unlist(mget(featureNames(gse.filt2), hgu133plus2ENTREZID))
anyDuplicated(G2.entrezid)

## [1] 0
```

Aplicamos el procedimiento de expresión diferencial utilizado en el ejemplo 7.2.

```
library(multtest)
gse.aov = rowFtests(gse.filt2, pData(gse20986)[,"tissue"])
p.originales = gse.aov[, 2]
p.BH = mt.rawp2adjp(p.originales, "BH")
pvalores = p.BH$adjp[p.BH$index, 2]
sig.1234 = which(pvalores < 0.001)
selected = unlist(mget(featureNames(gse.filt2[sig.1234,]), hgu133plus2ENTREZID))
```

Vamos a aplicar un test de Fisher unilateral para los grupos definidos de acuerdo con Gene Ontology. Cargamos paquetes necesarios.

```
library(GO.db);library(Category);library(GOstats)
```

Y realizamos los tests.

```
params = new("GOHyperGParams", geneIds = selected, universeGeneIds = G2.entrezid,
annotation = annotation(gse.filt2), ontology = "BP", pvalueCutoff = 0.01,
conditional = FALSE, testDirection = "over")
overRepresented = hyperGTest(params)
```

Finalmente para visualizar los resultados guardamos los resultados en un fichero y lo vemos con el navegador.

```
fl = tempfile()
htmlReport(overRepresented, file = fl)
browseURL(fl)
```

15.4.3 ALL

Esta sección se reproduce el análisis propuesto en http://www.bioconductor.org/help/course-materials/2009/SSCMay09/gsea/HyperG_Lecture.pdf. Utilizamos los datos [80, ALL]. En § 5.4 se indica cómo conseguir los datos `bcrneg` en donde hemos seleccionado algunas muestras. Los datos los tenemos en `bcrneg`. Veamos número de genes y muestras.

```
dim(bcrneg)

## Features  Samples
##    12625       79
```

Realizamos un filtrado no específico quitando aquellos cuyo rango intercuartílico esté por debajo de la mediana (de los rangos intercuartílicos observados). Además se requiere que el gen tenga anotación en Gene Ontology. Esto lo pedimos con el argumento `require.GOBP = TRUE`.[4]

```
bcrneg.filt = nsFilter(bcrneg, var.cutoff = 0.5, require.GOBP = TRUE)$eset
```

Veamos qué nos queda.

```
dim(bcrneg.filt)

## Features  Samples
##     4086       79
```

Vamos a generar una lista de genes significativos de un modo muy básico[5]. Calculamos el p-valor del test de la t y nos quedamos con los genes con un p-valor por debajo de 0.05.

```
fac0 = pData(bcrneg.filt)[, "mol.biol"]
fac0 = factor(fac0) ## Quitamos categorías vacías
rtt = rowttests(bcrneg.filt, fac0)
rttPrb = rtt$p.value
tThresh = rttPrb < 0.05
```

[4] En concreto se pide su anotación en la ontología *Biological Process*.
[5] Y tampoco muy recomendable.

Tabla 15.4: Tabla de contingencia para el término GO:0006468.

	InGO	
Selected	FALSE	TRUE
FALSE	3101	162
TRUE	658	30

Le damos nombres a las componentes del vector utilizando los identificadores de Affymetrix GeneChip.

```
names(rttPrb) = featureNames(bcrneg.filt)
```

Guardamos estos identificadores en `ids`.

```
ids = featureNames(bcrneg.filt)
```

Guardamos también las correspondencias con Entrez.

```
map = hgu95av2ENTREZID
```

Definimos quién es nuestro universo. En nuestro caso todos los genes que intervenían en nuestro estudio y de los cuales teníamos su anotación (hay sondas de control que desaparecen con el filtrado que acabamos de hacer).

```
universe = unlist(mget(ids, map))
selected = unlist(mget(ids[tThresh], map))
```

Elegimos un grupo utilizando un término de Gene Ontology. Primero cargamos el paquete [25, GO.db].

```
library(GO.db)
```

Nos fijamos por ejemplo en el término siguiente

```
GOTERM[["GO:0006468"]]

## GOID: GO:0006468
## Term: protein phosphorylation
## Ontology: BP
## Definition: The process of introducing a
##     phosphate group on to a protein.
## Synonym: protein amino acid phosphorylation
```

La tabla 15.4 muestra los conteos observados.
Cargamos los paquetes [55, Category] y [56, GOstats].

```
library(Category)
library(GOstats)
```

```
params = new("GOHyperGParams", geneIds = selected, universeGeneIds = universe,
annotation = annotation(bcrneg.filt), ontology = "BP", pvalueCutoff = 0.001,
conditional = FALSE, testDirection = "over")
overRepresented = hyperGTest(params)
```

Veamos el resumen.

```
head(summary(overRepresented), n = 3)
```

Guardamos los resultados en un fichero HTML y lo vemos.

```
fl = tempfile()
htmlReport(overRepresented, file = fl)
browseURL(fl)
```

15.5 Ejercicios

Ejercicio 30

Utilizamos los datos tamidata::gse20986. En el problema 21 hemos determinado los genes significativos con un FDR de 0.05 para las comparaciones entre las muestras obtenidas en el iris, retina y coroides con las muestras huvec. Tenemos pues tres grupos de genes significativos que podemos denotar S_{iris}, S_{retina} y $S_{coroides}$.

1. Realizar, para cada uno de los tres grupos de genes seleccionados, un análisis de un sobre solapamiento, utilizando el test de Fisher unilateral, con los grupos definidos en Gene Ontology. En concreto hay que utilizar las tres bases de bases de Gene Ontology: BP (procesos biológicos), CC (componentes celulares) y MP (función molecular).

Capítulo 16

Análisis de conjuntos de genes

16.1 Introduction

[1]

El problema abordado en este capítulo corresponde a lo que se conoce como análisis de conjunto de genes.[204] Estudiamos si hay relación entre conjuntos de genes **previamente definidos** y fenotipo.[205] Estos grupos vendrán definidos según distintos criterios. Por ejemplo, corresponder a una ruta metabólica, o localizados en un mismo cromosoma o bien definidos utilizando términos de Gene Ontology. De un modo genérico, un conjunto de genes que represente algo interpretable desde un punto de vista biológico. Ya no estamos interesados en un análisis *marginal* o *gen a gen* de los datos de expresión. En un análisis de expresión diferencial el resultado final es una lista ordenada de genes de modo que una mayor asociación con la covariable fenotípica produce una posición más alta en la lista. Un procedimiento de tests múltiples nos produce una clasificación en genes significativos y no significativos. En definitiva un valor de corte de la lista. En esta aproximación *no usamos ningún conocimiento previo sobre relaciones conocidas entre genes* que podrían ser esenciales a la hora de determinar la asociación no ya gen-fenotipo sino conjunto de genes-fenotipo.

[204] En la literatura se refieren a este problema como *gene set analysis*, *gene set enrichment analysis*, *set based enrichment analysis*.

[205] Donde fenotipo se entiende en un sentido amplio como en todo el texto.

16.2 Sobre la distribución de la matriz de expresión

Denotaremos (como siempre) la matriz de expresión (aleatoria) como

$$X = [X_{ij}]_{i=1,\ldots,N;j=1,\ldots,n}$$

[1] Lo tratado en este capítulo tiene mucho que ver con la obra teatral de Lope de Vega, Fuenteovejuna. Se recomienda el video http://www.youtube.com/watch?v=-IcuFn57nAo. - *¿Quién mató al Comendador?*
- Fuenteovejuna, señor.
-¿Quién es Fuenteovejuna?
- Todo el pueblo, a una. Esta frase aparece repetida en la obra y es básica en este tema. Este tema va de esto, de la acción conjunta de un conjunto de genes. Unos pocos habitantes no pueden pero todos los habitantes de Fuenteovejuna sí que pueden.

donde X_{ij} es la expresión aleatoria del i-ésimo gen en la j-ésima muestra. Las expresiones *observadas* serán

$$x = [x_{ij}]_{i=1,\ldots,N;j=1,\ldots,n}$$

Notemos que X_{ij} es una variable aleatoria mientras que x_{ij} es un valor observado. Las columnas de la matriz de expresión X son independientes. Son vectores aleatorios independientes[206] Por tanto podemos considerar que las distintas muestras son realizaciones de vectores independientes aunque no con la misma distribución ya que son observados bajo distintas condiciones experimentales. Si nos fijamos en las filas de la matriz de expresión, esto es, en los perfiles de expresión entonces tenemos realizaciones de vectores aleatorios dependientes y que además no tienen la misma distribución. Los genes no son independientes en su comportamiento hay corregulaciones entre ellos. Además tampoco tienen porqué comportarse de un modo (aleatorio) común.

[206] Corriendo un tupido velo por toda la parte de preprocesado de la información que hemos visto en § 4 para datos de microarrays. Es claro que el preprocesado de la información introduce dependencias entre valores observados para distintas muestras pero las ignoramos.

16.3 Conjunto(s) de genes

Tenemos distintos conjuntos de genes previamente definidos utilizando información previa: un grupo(s) definido por el grupo de investigación, grupos definidos utilizando Gene Ontology, ... Lo fundamental, estos grupos de genes no han de estar definidos en función de *nuestros* datos de expresión.[207] Si denotamos por $G = \{1, \ldots, N\}$ el conjunto total de genes considerado (o universo de genes) entonces los conjuntos de genes serán $S_1, \ldots, S_K$. En particular, no es extraño considerar el caso $K = 1$, es decir, estar interesados en un conjunto de genes dado. Supondremos que el conjunto S_k tiene cardinal n_{S_k} o, si no hay confusión, simplemente n_k. Los distintos conjuntos S_k **no** son una partición de G: no son necesariamente disjustos ($S_i \cap S_j \neq \emptyset$ para $i \neq j$) y su unión no tiene porqué ser todo el universo considerado de genes.

[207] § 14.

La cuestión básica podemos formularla como: *¿Hay asociación entre un conjunto de genes y el fenotipo?*[208] Es una pregunta muy vaga. Se requiere una formulación más precisa. Caben distintas interpretaciones de la pregunta. En [132] formulan las siguientes dos hipótesis nulas que concretan de dos modos distintos la cuestión previa. Reproducimos las hipótesis nulas.[209]

[208] Esto es, la misma pregunta que nos formulábamos para cada gen pero ahora referido a un conjunto dado de genes.

[209]

Hipótesis Q1: Los genes de un conjunto muestran el mismo patrón de asociación con el genotipo comparado con el resto de genes.

Hipótesis Q2: El conjunto de genes no tiene ningún gen cuyo nivel de expresión está asociado con el fenotipo de interés.

Hypothesis Q1: The genes in a gene set show the same pattern of associations with the phenotype compared with the rest of the genes.

Hypothesis Q2: The gene set does not contain any genes whose expression levels are associated with the phenotype of interest.

No tenemos la misma hipótesis nula, no. La hipótesis nula Q1 se centra en la comparación entre (la asociación entre) un conjunto dado de genes (con el fenotipo) y (la asociación entre) los otros (con el fenotipo). En cambio, la hipótesis Q2 se centra en el estudio de la expresión diferencial de los genes que pertenecen a un conjunto dado de genes.

Un planteamiento similar lo podemos encontrar en [59]. En concreto un test que se plantea si un conjunto de genes tiene una asociación con el fenotipo, la hipótesis Q2, recibe el nombre de *test autocontenido* (*self-contained test*) mientras que si nos ocupamos de la hipótesis

nula Q1 hablamos de un *test competitivo* (*competitive test*). De hecho el formula las hipótesis nulas del siguiente modo:

Hipótesis nula competitiva H_0^{comp}: Los genes en un grupo dado S están como mucho tan frecuentemente expresados de un modo diferencial como los genes en S^c.

Hipótesis nula autocontenida H_0^{auto}: Ningún gen en S está diferencialmente expresado.

[2]

Muchos procedimientos estadísticos propuestos contrastar estas hipótesis no formula expresamente cuales son las hipótesis nulas que están contrastando. De hecho, la reflexión sobre el propio procedimiento de contraste propuesto es el que nos ha de indicar la hipótesis nula. No tenemos un modelo (estocástico) preciso y esto ocasiona esta indeterminación.

En lo que sigue veremos procedimientos que asumen una distribución conocida (o al menos asintóticamente conocida) para los estadísticos de contraste. Sin embargo, la opción más utilizada es evaluar la significación del estadístico asociado al contraste utilizando como distribución nula (o distribución bajo la hipótesis nula) una distribución de aleatorización o una distribución bootstrap. Qué distribución es adecuada dependerá de la hipótesis que estemos contrastando. De hecho, desde el punto de vista estadístico, este es el núcleo del problema. ¿Cómo estimamos la distribución nula?

16.4 Ejemplos

En esta sección comentamos los ejemplos que utilizamos.

16.4.1 Un ejemplo simulado de Efron y Tibshirani

En [43] se proponen un par de ejemplos con datos simulados. Son los ejemplos 16.1 y 16.2.

Ejemplo 16.1 *Se consideran 1000 genes y 50 muestras. Se supone que las 25 primeras muestras son controles y las últimas 25 es el grupo de tratamiento. Definimos los grupos de genes como: las 20 primeras filas (de la matriz de expresión) corresponden al primer grupo, de la 21 a la 40 el segundo y así sucesivamente. Los niveles de expresión los generamos aleatoriamente independientemente y con la misma distribución. La distribución común es una normal estándar ($X_{ij} \sim N(0,1)$). En el primer grupo añadimos un valor constante de 2.5 a los primeros 10 genes en las muestras correspondientes a tratamiento (últimas 25 columnas de la matriz de expresión). En consecuencia, lo que hacemos es que solamente el primer grupo tiene la mitad de los genes asociados con el fenotipo y la otra mitad sin ningún tipo de asociación. El resto de grupos no tiene ninguna asociación con fenotipo. Generamos estos datos.*

[2] Es interesante leer las hipótesis nula tal como las formulan:

Competitive null hypothesis H_0^{comp}: The genes in S are at most as often diferentially expressed as the genes in S^c.

Self-contained null hypothesis H_0^{self}: No genes in S are differentially expressed.

```
N = 1000; n = 50
set.seed(280562) ## Para obtener los mismos valores generados
et1  = matrix(rnorm(N*n),nrow = N,ncol = n)
et1[1:10,26:50] = et1[1:10,26:50] + 2.5
```

En este ejemplo el vector que nos indica la clasificación de las muestras será

```
y.et = factor(rep(1:2,rep(25,2)),levels = 1:2,
  labels = c("Normal","Tratamiento"))
```

En lo que sigue utilizaremos como medida de asociación fenotipo-expresión el estadístico del t-test.

```
et1.tt = genefilter::rowttests(et1,y.et)$statistic
```

En figura 16.2 tenemos una estimación de la densidad. Podemos ver cómo se aprecia, por debajo de 8, el valor que corresponde al primer grupo.

```
pacman::p_load("ggplot2")
df = data.frame(et1.tt)
png(paste0(dirTamiFigures,"GeneSetAnalysis5b.png"))
ggplot(df,aes(x=et1.tt))+geom_density()
dev.off()

## pdf
##   2
```

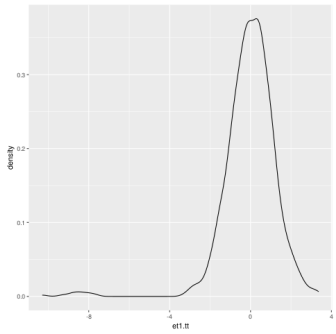


Figura 16.1: Densidad estimada de et1.tt.

Ejemplo 16.2 *Este ejemplo se define exactamente como el ejemplo 16.1 excepto lo que se hacía solamente para el primer grupo lo hacemos para todos: sumamos a los 10 primeros genes de cada grupo 2.5 unidades en las muestras correspondientes al tratamiento (últimas 25 columnas de la matriz de expresión). Generamos los datos.*

```
N = 1000; n = 50
set.seed(280562)
indices.temp  = (0:49)*20 + 1
indices = NULL
for(i in indices.temp) indices = c(indices,i:(i+9))
et2  = matrix(rnorm(N*n),nrow = N,ncol = n)
et2[indices,26:50] = et1[1:10,26:50] + 2.5
```

Notemos que la covariable indicando el grupo es la misma que en ejemplo 16.1. Definimos los grupos para su uso posterior.

```
gen.name = function(i) paste("g",((i-1)*20 +1):(i*20),sep="")
gsc.et = lapply(as.list(1:50),gen.name)
names(gsc.et) = paste("set",as.character(1:50),sep="")
gsnames.et = paste("set",as.character(1:50),sep="")
genenames.et = paste("g",1:1000,sep="")
```

```
et2.tt = genefilter::rowttests(et2,y.et)$statistic
```

En figura 16.2 tenemos una estimación de la densidad.

```
df = data.frame(et2.tt)
p = ggplot(df,aes(x=et2.tt))+geom_density()
```

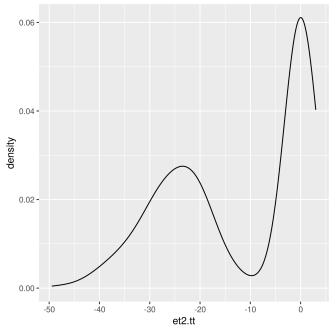


Figura 16.2: Densidad estimada de et1.tt.

16.5 Cuantificando asociación gen-fenotipo

El primer paso es cuantificar la asociación entre los niveles de expresión de cada gen y el fenotipo. Esta cuantificación será un estadístico (una función de los datos que son las expresiones en la fila de la matriz de expresión) cuya definición dependerá del tipo de información fenotípica disponible. Por ejemplo, si tenemos dos condiciones entonces este estadístico será (habitualmente pero no siempre) el estadístico t que se utiliza en la comparación de medias de dos poblaciones normales. Pero no necesariamente, por ejemplo, si los tamaños muestras n_1 y n_2 son muy pequeños entonces muy probablemente un estadístico como la diferencia de medias es suficiente (y menos peligroso).

Si las muestras las tenemos clasificadas en más de dos grupos entonces podemos utilizar el valor del estadístico F del análisis de la varianza (§ 7.8).

En § 9.1 se estudiaba el método SAM (en el contexto de la expresión diferencial marginal). Este método propone, dependiendo de las descripción fenotípica con la que se trabaja, un estadístico d_i para cada gen. La definición de este estadístico es función del fenotipo. Son perfectamente utilizables como medidas de asociación gen-fenotipo (es lo que son).

En lo que sigue a la medida de asociación gen-fenotipo la denotaremos genéricamente por t_i (aunque no sea un t estadístico).

Figura 16.3: Un anuncio.

16.6 Enriqueciendo el conjunto de genes

El vector $\boldsymbol{t} = (t_1, \ldots, t_N)$ constituye, de hecho, una descripción de la asociación *marginal* gen-fenotipo. Pero tenemos unos grupos de genes en los que tenemos interés o que expresan simplemente conocimiento previo de posibles asociaciones entre los genes. Sea S uno de estos conjuntos. Enriquecer el estadístico t para el conjunto consiste en considerar un resumen de t sobre S. Y esto lo podemos hacer de muchas formas. Si denotamos el estadístico de enriquecimiento por $t(S)$ la opción más simple es

$$\boldsymbol{t}(S) = \sum_{i \in S} \frac{t_i}{n_S}, \tag{16.1}$$

es decir, el promedio de los t_i observados sobre el conjunto de genes de interés S. Podemos considerar muchas otras opciones y en las secciones que siguen las veremos.

16.7 Distribuciones condicionadas a los datos

Sea x la matrix de expresión observada. Y supongamos que el vector y (de dimensión n) nos indica la covariable de interés de la muestra (normalmente la pertenencia a un grupo). Para el universo de genes considerado tendremos el el vector $\boldsymbol{t} = (t_1, \ldots, t_N)$.

16.7.1 Distribución de permutación para un gen

Para un gen tenemos su perfil de expresión $u = (u_1, \ldots, u_n)$ y tenemos el vector y asociado a las muestras.

Sea π denota una permutación aleatoria de $(1, \ldots, n)$ de forma que los $\pi(i)$ será el índice que ocupa la posición i-ésima en la permutación π. En particular denotaremos por π_0 la permutación que nos devuelve el orden original: $\pi_0(j) = j$. Si el vector asociado a las muestras es y entonces tendremos el vector (permutado de y) $y_\pi = (y_{\pi(1)}, \ldots, y_{\pi(n)})$. Obviamente $y = y_{\pi_0}$.

La cantidad que describe la asociación entre u y y_π es t_π. Utilizando esta notación el valor observado de la asociación gen-fenotipo para u e y será t_{π_0}. Si consideramos B permutaciones aleatorias entonces tendremos los valores $t_{\pi_1}, \ldots, t_{\pi_B}$ para las B permutaciones aleatorias.

Si no hay asociación gen-fenotipo entonces el valor de t_{π_0} debe de ser *como* los valores $t_{\pi_1}, \ldots, t_{\pi_B}$. De hecho, cualquier ordenación de $t_{\pi_0}, t_{\pi_1}, \ldots, t_{\pi_B}$ tiene la misma probabilidad. Podemos considerar dos casos. En el primero una mayor asociación se expresa como un valor mayor de t (normalmente positivo). ¿Cuántos valores t_{π_b} (con $b = 1, \ldots, B$) son mayores que t_{π_0}? Si hay pocos indica que no hay equiprobabilidad y, por lo tanto, rechazamos esa hipótesis. Esto es un test de aleatorización aplicado a las muestras. El p-valor sería la proporción de t_{π_b}s mayores que t_{π_0}, es decir,

$$p_r = \frac{|\{b : t_{\pi_b} > t_{\pi_0}\}|}{B}. \tag{16.2}$$

En el caso en que una mayor asociación se exprese como un valor muy grande (positivo) o muy pequeño (negativo) de t entonces el p valor vendría dado por

$$p_r = \frac{|\{b : |t_{\pi_b}| > |t_{\pi_0}|\}|}{B}, \tag{16.3}$$

ya que tendríamos un test bilateral.

Ejemplo 16.3 *Consideremos el ejemplo 16.1, en concreto, los valores del primer gen. La covariable y nos indica la pertenencia a control o tratamiento. Como medida de asociación consideramos el estadístico t (con varianzas desiguales). Notemos que una asociación grande supone un valor o muy grande (positivo) o muy pequeño (negativo).*

```
u = et1[1,]
t0 = t.test(u ~ y.et)$statistic
t0 = abs(t0)
```

Generamos $B = 100$ permutaciones aleatorias de y utilizando la función `sample`.

```
B = 100
tb = rep(0,B)
for(i in 1:B) tb[i] = t.test(u ~ sample(y.et))$statistic
```

Determinamos los valores absolutos de los estadísticos.

```
tb = abs(tb)
```

El valor observado de t_{π_0} es 8.77. Y el p-valor será la proporción de los que su valor absoluto es mayor que el valor absoluto de t_{π_0}), es decir,

```
sum(tb > t0) / B

## [1] 0
```

Tenemos un p-valor nulo. No parece *que todas las ordenaciones de los valores $t_{\pi_0}, t_{\pi_1}, \ldots, t_{\pi_B}$ sean equiprobables.*

La distribución nula que hemos considerado en esta sección corresponde con la hipótesis nula autocontenida que veíamos previamente. No asumimos ninguna distribución asintótica para el estadístico enriquecido y generamos una distribución nula en la que intercambiamos las etiquetas de las muestras. Si nuestro interés es contrastar la hipótesis Q2 o hipótesis autocontenida esta sería la distribución nula natural.[210]

[210] Podemos hablar de aleatorización por columnas entendiendo que las distintas muestras corresponden a las distintas columnas en la matriz de expresión.

16.7.2 Distribución de aleatorización

Denotemos por S_0 el conjunto de genes original en el que tenemos interés. Ahora vamos a elegir al azar un grupo S de genes (filas) del mismo cardinal que S_0. Lo aleatorio ahora no es el vector y sino el conjunto S de genes. Si S es aleatorio entonces también lo es $\boldsymbol{t}(S)$ (ahora las muestras tienen su ordenación original). A la distribución de probabilidad de $\boldsymbol{t}(S)$ se le llama **distribución de aleatorización** del estadístico de enriquecimiento. Si tomamos B selecciones aleatorias de n_{S_0} genes tendremos los conjuntos S_b con $b = 1, \ldots, B$ y los valores del estadístico de enriquecimiento observados son $\boldsymbol{t}(S_0)$ (para el grupo original) y $\boldsymbol{t}(S_b)$ con $b = 1, \ldots, B$ para los seleccionados al azar. El p-valor se define análogamente como

$$p = \frac{|\{b : t(S_b) > t(S_0)\}|}{B}. \tag{16.4}$$

si solamente rechazamos para valores grandes (positivos). En el caso bilateral tendremos

$$p = \frac{|\{b : |t(S_b)| > |t(S_0)|\}|}{B}. \tag{16.5}$$

La distribución nula que hemos considerado en esta sección corresponde con la hipótesis nula competitiva o hipótesis Q1. No asumimos ninguna distribución asintótica para el estadístico enriquecido y generamos una distribución nula en la que seleccionamos al azar grupos del mismo tamaño. Realmente lo que estamos haciendo es generar permutaciones aleatorias de las filas en la matriz de expresión. Si nuestro

interés es contrastar la hipótesis Q1 o hipótesis competitiva esta sería la distribución nula natural.[211]

En lo que sigue repasamos distintos paquetes R/Bioconductor que implementan distintas medidas de enriquecimiento y distribuciones nulas en donde permutamos aleatoriamente las columnas (muestras) o las filas (características).

[211] Podemos hablar de aleatorización por filas entendiendo que los distintos genes (o genéricamente características que observamos) corresponden a las distintas filas en la matriz de expresión. También se habla de muestreo de genes o *gene sampling*.

16.8 Limma::genSetTest

En el paquete [126, limma] tenemos la función limma::geneSetTest. En este caso la medida de enriquecimiento que se utiliza es la media muestra de los estadísticos calculados para cada gen. Simplemente se permuta por filas. Contrastamos pues la hipótesis competitiva.

Ejemplo 16.4 (genSetTest y ejemplo 16.1) *Consideremos los datos del ejemplo 16.1 y el t-estadístico para cada gen. El grupo de interés es el primero (primeras 20 filas). Supongamos que tomamos como alternativa si tienden a tomar valores mayores en el primer grupo.*

```
pacman::p_load(limma)
geneSetTest(1:20,et1.tt,alternative ="up")

## [1] 0.9984722
```

El p-valor nos indica que no rechazamos la hipótesis nula. ¿Y si contrastamos la alternativa de valores mayores en el segundo grupo?

```
geneSetTest(1:20,et1.tt,alternative ="down")

## [1] 0.00153168
```

Vamos a realizar el contraste para cada uno de los 50 grupos considerados.

```
pvalores = NULL
for(i in 0:49){
  indices = (i * 20 + 1): (i * 20 + 10)
  p.temp = geneSetTest(indices,et1.tt,alternative ="down")
  pvalores = c(pvalores,p.temp)
}
```

Comprobamos que el p-valor del primer grupo es claramente menor que el p-valor para los restantes grupos.

16.9 Limma::wilcoxGST

Siguiendo con el paquete [126, limma] una opción simple (no necesariamente muy potente) y que no utiliza la distribución de aleatorización consiste en tomar los valores del estadístico en el grupo de interés y en el resto de genes y compararlos utilizando un test de Wilcoxon. Por lo tanto estamos en el caso en que **conocemos** la distribución nula. No aleatorizamos ni por filas ni por columnas. Es un test exacto. El test de Wilcoxon tampoco asume ninguna hipótesis distribucional

sobre los estadísticos por gen que utilizamos lo que es una ventaja. Sin embargo, asume independencia entre estos estadísticos que **no** se verifica ya que las expresiones de los genes son interdependientes.

Ejemplo 16.5 (Datos de ejemplo 16.1) *Utilizamos la función limma::wilcoxGST.*

```
wilcoxGST(1:20,et1.tt,alternative ="down")

## [1] 0.00153168

wilcoxGST(1:20,et1.tt,alternative ="up")

## [1] 0.9984722

wilcoxGST(1:20,et1.tt,alternative ="mixed")

## [1] 2.837697e-08
```

Tomemos otro grupo y repitamos el análisis.

```
wilcoxGST(21:30,et1.tt,alternative ="down")

## [1] 0.09546351

wilcoxGST(21:30,et1.tt,alternative ="up")

## [1] 0.904723

wilcoxGST(21:30,et1.tt,alternative ="mixed")

## [1] 0.8181643
```

16.10 Limma::CAMERA

El método fue propuesto en [148].[212] Es un procedimiento para contrastar la hipótesis competitiva. Es un procedimiento que tiene en cuenta la correlación entre las expresiones de los distintos genes en cada muestra. La mayor parte de los procemientos propuestos para el contraste de la hipótesis competitiva suelen asumir (falsamente) la independencia de la expresión observada para distintos genes y su validez depende una hipótesis que sabemos no es cierta. Se ha visto que no tener en cuenta esta dependencia entre genes nos lleva a un incremento de la tasa de error FDR.

Se asume que la expresión está cuantificada en escala logarítmica (base 2 como es habitual en este contexto).

$$E[X_{ij}] = \mu_{ij} = \sum_{k=1}^{p} \alpha_{ik} y_{jk} \tag{16.6}$$

siendo y_{jk} la k-ésima covariable (o variable fenotípica) de la j-ésima muestra. Se supone que las expresiones aleatorias (su perfil aleatorio de expresión) de un mismo gen tienen una varianza común σ_i^2.[213] Se asume que las expresiones para distintas muestras son independientes pero no entre distintos genes. En particular, denotamos el coeficiente

[212] El nombre es un acrónimo **C**orrelation **A**djusted **ME**an **RA**nk.

[213] Asumible si hemos aplicado previamente algún procedimiento de normalización.

de correlación de Pearson entre las variables aleatorias $X_{i_1 j}$ y $X_{i_2 j}$ como $cor(X_{i_1 j}, X_{i_2 j}) = \rho_{i_1,i_2}$. Obviamente no asumimos que estos coeficientes sean nulos.[214]

[214] No asumimos incorrelación. Recordemos que independencia implica incorrelación pero no viceversa.

Un contraste, para el gen i-ésimo viene dado por

$$\beta_i = \sum_{k=1}^{p} c_j \alpha_{ik}.$$

Se está interesado en contrastar la hipótesis nula de que el contraste es nulo. Tenemos interés en los contrastes de hipótesis: $H_0 : \beta_i = 0$ frente a la alternativa de que $H_0 : \beta_i \neq 0$. Denotemos el estadístico del contraste como Z_i.[215] ¿Qué estadísticos Z_i vamos a considerar?

[215] No necesariamente con distribución normal.

16.11 GSA

[216] En el paquete [42, GSA] se implementa el método propuesto en [43].

[216] Se utiliza la distribución de permutación a la que aplican una reestandarización. La medida de enriquecimiento que proponen por defecto [3] es la siguiente: partimos de los valores t_i que miden asociación gen-fenotipo. Definimos $t^+ = \max\{t, 0\}$ y $t^- = -\min\{t, 0\}$. Consideramos un conjunto de genes S. Definimos $\bar{t}_S^+ = \sum_{i \in S} \frac{t_i^+}{n_S}$ y $\bar{t}_S^- = \sum_{i \in S} \frac{t_i^-}{n_S}$. Finalmente la medida de enriquecimiento (que llamaremos el estadístico `maxmean`) es

$$\boldsymbol{t}(S) = \max\{\bar{t}_S^+, \bar{t}_S^-\}. \tag{16.7}$$

Estamos tomando la media de las partes positivas t_i^+, la media de las partes negativas t_i^- y nos quedamos con el máximo de ambos valores.

Ejemplo 16.6 *[43, página 119] La medida de enriquecimento que acabamos de definir es robusta frente al caso en que tengamos una medida extrema. El ejemplo que proponen los autores es el siguiente: supongamos que tenemos 100 genes, 99 de los valores t_i's son -0.5 y el valor restante es 10. Tenemos que $\bar{t}_S^+ = 10/100 = 0.1$ y $\bar{t}_S^- = -99(-0.5)/100 = 0.495$. Vemos que los valores negativos dominan pues son la mayor parte de los datos. Si se tomara la media de las partes positivas t^+ y la media de las partes negativas t^- tendríamos los valores 10 y -0.5 (es decir, solamente consideramos cuando el valor no es nulo) y la medida de enriquecimiento vendría dominada por valores extremos.*

Como medidas de asociación gen-prototipo t_i utilizan las mismas del paquete [133, samr] que aparecen en la sección § 9.1.1. Cuando analizan los conjuntos de genes hablan de *grupos de genes positivos* y *grupos de genes negativos*. Un grupo de genes se dice negativo si corresponden con genes que en la clase 2 tiene expresiones menores cuando tenemos dos grupos (1 y 2). Si tenemos una covariable y numérica entonces los negativos corresponden al caso en que expresiones menores se asocian a valores mayores de y. Los grupos positivos se definen de modo contrario a los positivos. Cargamos el paquete.

```
pacman::p_load(GSA)
```

[3] Aunque lleva el promedio de los t_i's y el promedio de $|t_i|$ como otras opciones.

Ejemplo 16.7 (Ejemplo 16.1 con GSA) *Empezamos analizando el ejemplo 16.1.*

```
grupo = c(rep(1,25),rep(2,25)) #El grupo tiene que numerarse 1,2
et1.gsa = GSA(et1,grupo, genenames=genenames.et, genesets=gsc.et,
  resp.type="Two class unpaired", nperms=100)
```

Podemos ver los estadísticos enriquecidos para los grupos

```
head(et1.gsa$GSA.scores)

## [1]  3.99534187 -0.01231818  0.11759218
## [4] -0.27081919  0.05557714 -0.07898232
```

El valor observado para el grupo 1 es 4 mientras que una descriptiva de los demás es la siguiente

```
summary(et1.gsa$GSA.scores[-1])

##      Min.   1st Qu.    Median      Mean   3rd Qu.
## -0.44625 -0.12429 -0.04239 -0.05241  0.05558
##      Max.
##  0.21225
```

Los p-valores obtenidos para lo que ellos llaman genes negativos *que corresponden con genes que en la clase 2 tiene expresiones menores. Podemos obtenerlos con*

```
head(et1.gsa$pvalues.lo)

## [1] 1.00 0.41 0.55 0.07 0.62 0.32
```

Nuestro grupo 1 no destaca. Si considerados los p-valores de grupos positivos (en grupo 2 expresiones mayores) tenemos

```
head(et1.gsa$pvalues.hi)

## [1] 0.00 0.59 0.45 0.93 0.38 0.68
```

que para el grupo 1 vale 0 y un resumen de los demás es

```
summary(et1.gsa$pvalues.hi[-1])

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1500  0.4500  0.5800  0.5776  0.7300  0.9700
```

Los valores originales t_i los tenemos con

```
head(et1.gsa$gene.scores)

## [1] 3.485459 3.901185 3.698664 4.143949 3.555439
## [6] 3.656246
```

En la figura 16.4 podemos ver un diagrama de cajas que compara el primer grupo con los demás.

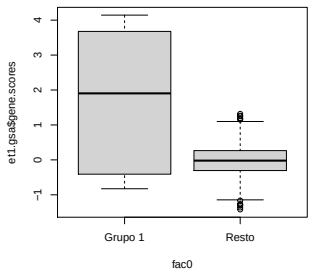


Figura 16.4: Valores t_i para el primer grupo (izquierda) y los demás. El primer grupo tiene diez genes claramente diferenciados mientras que los demás no lo están. De ahí la gran variabilidad que muestra el grupo.

```
fac0 = factor(c(rep(1,20),rep(2,980)),levels = 1:2,labels=c("Grupo 1","Resto"))
boxplot(et1.gsa$gene.scores ~ fac0)
```

En el primer grupo tenemos la mitad de los genes diferenciados y la otra mitad no. En el resto de los genes no hay diferenciación. De ahí la gran variabilidad del primer grupo y que se solape por la parte inferior con el resto de los genes. El análisis de grupo que hemos realizado lo diferencia sin problemas.

En la figura 16.5 tenemos los grupos de genes que se considerarían significativos (diferenciando grupos up y down) y el valor de FDR para que los declaremos significativos en los datos `et1` *(ejemplo 16.1).*

```
GSA.plot(et1.gsa)
```

Vemos cómo solamente admitimos un grupo con un valor bajo de FDR.

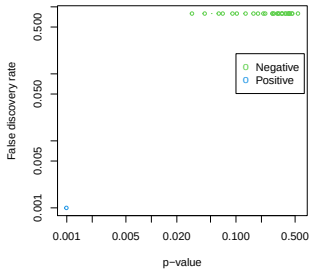


Figura 16.5: Grupos significativos con datos et1 en función de FDR.

Ejemplo 16.8 (Ejemplo 16.2 con GSA) *En el ejemplo 16.2 todos los grupos considerados tienen el mismo comportamiento. Aunque todos tienen expresión diferencial, no hay un comportamiento diferenciado del grupo respecto de los otros grupos. Los datos son* `et2` *mientras que los grupos los tenemos en* `gsc.et` *con nombres en* `gsnames.et`*. Finalmente* `genenames.tt` *nos da los nombres de los genes.*

```
grupo = c(rep(1,25),rep(2,25))
et2.gsa = GSA(et2,grupo, genenames=genenames.et, genesets=gsc.et,
    resp.type="Two class unpaired", nperms=100)
```

La figura 16.6 muestra los grupos significativos en función de la FDR para los datos `et2` *(ejemplo 16.2).*

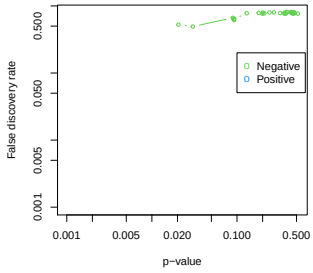


Figura 16.6: Grupos significativos con datos et2 en función de FDR.

```
GSA.plot(et2.gsa)
```

Es claro que ningún grupo se diferencia de los demás.

Ejemplo 16.9 (Datos GSE1397 y GSA) *Leemos los datos.*

```
data(gse1397,package="tamidata")
data(gse1397.gsc,package="tamidata")
gsc = gse1397.gsc
gruposGrandes = which(sapply(geneIds(gsc),length) > 50)
gsc = gsc[gruposGrandes]
gse = gse1397
```

Realizamos el análisis. Previamente convertimos la variable `tipo` *que es un factor a un vector numérico con valores 1 y 2 (es como lo pide [42, GSA].*

```
tipo.num  = as.numeric(pData(gse)[,"type"])
```

Utilizamos como nombre de los genes los AffyId.

```
gse1397.gsa = GSA(exprs(gse),tipo.num, genenames=featureNames(gse),
  genesets=geneIds(gsc),resp.type="Two class unpaired", nperms=1000)
```

En la figura 16.7 podemos ver la representación de la tasa de falsos positivos como función del p-valor.

```
GSA.plot(gse1397.gsa)
```

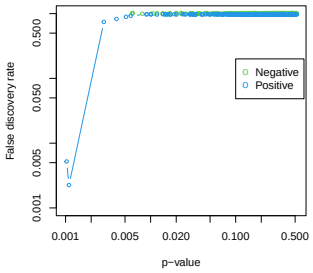


Figura 16.7: Grupos significativos con datos gse1397.

Fijamos una tasa de error de 0.05. Veamos qué grupos son los que o bien con una asociación negativa o bien con asociación positiva son los que presentan una mayor diferenciación entre los dos grupos considerados (con y sin síndrome de Down). Primero determinar los índices de los grupos en nuestra colección.

```
(ind.lo = which(gse1397.gsa$pvalues.lo <.05))

##  [1]    7    8   38   56  103  105  119  138  180
## [10]  220  257  262  281  351  356  418  426  438
## [19]  454  459  478  494  543  566  569  575  608
## [28]  610  613  655  684  857  910  972 1011 1012
## [37] 1146 1161

(ind.hi = which(gse1397.gsa$pvalues.hi <.05))

##  [1]    2   44   45   67   69  174  177  239  263
## [10]  302  337  338  376  403  431  479  532  539
## [19]  582  589  619  635  636  680  697  712  714
## [28]  715  784  786  796  893  928  936  958 1055
## [37] 1080 1098 1102 1123 1149 1151 1162
```

Podemos ver sus (primeros) identificadores en Gene Ontology. Para unos

```
head(names(gsc[ind.lo]))

##   GO:0000002   GO:0000012   GO:0000018
## "GO:0000027" "GO:0000028" "GO:0000160"
##   GO:0000019   GO:0000022   GO:0000023
## "GO:0000271" "GO:0000462" "GO:0000466"
```

y análogamente

```
head(names(gsc[ind.hi]))

##   GO:0000002   GO:0000012   GO:0000018
## "GO:0000012" "GO:0000186" "GO:0000187"
##   GO:0000019   GO:0000022   GO:0000023
## "GO:0000302" "GO:0000305" "GO:0001507"
```

Y ahora viene el trabajo del especialista para ver hasta qué punto lo que sale tiene sentido o no.

16.12 GSEA: Gene set enrichment analysis

[217]

[217] Este método no está implementado en un paquete R. Los autores proporcionan código en R pero no en forma de paquete. Existe un interfaz Java para utilizarlo fuera de R/Bioconductor que no es

La referencia básica es [130] que presenta una versión modificada del procedimiento originalmente propuesto en [93]. Es el método más popular para realizar un enriquecimiento (agregado). Se basa en el uso del estadístico de Kolmogorov-Smirnov para dos muestras.[218] De hecho, utiliza una versión modificada de este estadístico que lo hace más sensible a la detección de interacción entre grupo de genes y fenotipo.

[218] En la sección § 7.6 se explica el método.

Método

Empezamos ordenando el universo de genes de acuerdo al grado de asociación gen-fenotipo utilizando algunos de los estadísticos propuestos en § 16.5. Tendremos la lista de genes ordenada, L.

Entradas

1. La matriz de expresión X con N filas y n columnas.
2. Procedimiento de ordenación con objeto de producir la lista ordenada de genes, L.
3. Un valor p
4. Un conjunto de genes S.

Cálculo del enriquecimiento

1. Calculamos medida de asociación gen-fenotipo, t_i.
2. Ordenamos el universo de genes de acuerdo a las medidas de asociación del paso 1. Denotamos los índices ordenados con $r_1 \ldots, r_N$, es decir,

$$t_{r_1} \geq \ldots \geq t_{r_N}.$$

3. Calculamos para cada i con $i = 1, \ldots, N$

$$h_S(i) = \frac{1}{\sum_{r_i \in S} |t_i|^p} \sum_{j \leq i; r_j \in S} |t_j|^p, \tag{16.8}$$

Calculamos también

$$m_S(i) = \sum_{j \leq i; r_j \notin S} \frac{1}{N - n_S}, \tag{16.9}$$

El valor de e_S es la máxima desviación de cero de $h_S(i) - m_S(i)$, es decir,

$$e_S = \max_{1 \leq i \leq N} |h_S(i) - m_S(i)|. \tag{16.10}$$

Si elegimos S de un modo aleatorio (del universo de genes) entonces e_S tendrá un valor pequeño en relación a lo que se observa cuando S no es aleatorio bien concentrándose en la parte superior de la lista o en la inferior o siguiendo algún patrón no aleatorio. Si $p = 0$ entonces e_S es el estadístico del test de Kolmogorov-Smirnov (ver sección § 7.6).

Estimación del p-valor

1. Consideramos una asignación aleatoria del fenotipo a las muestras (una permutación aleatoria de las muestras manteniendo fijo el fenotipo), reordenamos los genes y calculamos el valor $E(S)$.
2. Se repite el paso anterior un gran número de veces.

3. Si e_0 es el valor de e_S sobre los datos originales y $e_1, \ldots, e_B$ son los valores de e_S para las asignaciones aleatorias de fenotipo a muestras entonces el p-valor viene dado por

$$p = \begin{cases} 2\frac{|\{e_i : e_i \geq e_0\}|}{B}, & \text{for } e_0 > 0, \\ 2\frac{|\{e_i : e_i \leq e_0\}|}{B} & \text{for } e_0 < 0. \end{cases}$$

Contrastes múltiples Suponemos que consideramos ahora una colección de conjuntos de genes de interés: $S_1, \ldots S_K$.

1. Calculamos e_{S_k} para $k = 1, \ldots, K$.
2. Para cada S_k y 1000 permutaciones fijas π_i con $i = 1, \ldots, 1000$ (las mismas para todos los conjuntos de genes) del fenotipo, reordenamos los genes y calculamos el enriquecimiento e_{S_k,π_i}.
3. Ajustamos por el tamaño variable de los conjuntos de genes. Para ello consideramos los valores

$$\bar{e}_+ = \sum_{k,i:e_{S_k,\pi_i}>0} \frac{e_{S_k,\pi_i}}{|\{k,i : e_{S_k,\pi_i} > 0\}|},$$

y

$$\bar{e}_- = \sum_{k,i:e_{S_k,\pi_i}<0} \frac{e_{S_k,\pi_i}}{|\{k,i : e_{S_k,\pi_i} < 0\}|}.$$

Consideramos los valores

$$\tilde{e}_0 = \begin{cases} \frac{e_0}{\bar{e}_+}, & \text{si } e_0 > 0, \\ \frac{e_0}{\bar{e}_-}, & \text{si } e_0 < 0. \end{cases}$$

y

$$\tilde{e}_{S_k,\pi_i} = \begin{cases} \frac{e_{S_k,\pi_i}}{\bar{e}_+}, & \text{si } e_{S_k,\pi_i} > 0, \\ \frac{e_{S_k,\pi_i}}{\bar{e}_-}, & \text{si } e_{S_k,\pi_i} < 0. \end{cases}$$

4. Consideremos un valor normalizado (según método de punto anterior) $\tilde{e}$. Se *estima* la tasa de falsos positivos para ese valor como: si $\tilde{e} > 0$

$$q(\tilde{e}) = \frac{|\{k,i : e_{S_k,\pi_i} \geq \tilde{e}\}|/|\{k,i : e_{S_k,\pi_i} \geq 0\}|}{|\{k : e_{S_k} \geq \tilde{e}\}|/|\{k : e_{S_k} \geq 0\}|}$$

Si $\tilde{e} < 0$ entonces

$$q(\tilde{e}) = \frac{|\{k,i : e_{S_k,\pi_i} \leq \tilde{e}\}|/|\{k,i : e_{S_k,\pi_i} \leq 0\}|}{|\{k : e_{S_k} \leq \tilde{e}\}|/|\{k : e_{S_k} \leq 0\}|}$$

16.13 Estudiando interacciones entre genes individuales y grupos de genes

Suponemos que tenemos bien definido un grupo de genes S y pretendemos estudiar la posible asociación entre un gen g no incluido en S y el grupo S. Se trata de estudiar posibles asociaciones no conocidas previamente entre la actividad del gen individual y la actividad conjunta del grupo S.

En [21] consideran el caso en que las muestras están clasificadas en dos grupos. Se propone el siguiente método. Consideramos la matriz

de datos $[x_{ij}]_{i\in S, j\in\{1,\ldots,n\}}$. Se aplican una componentes principales a esta matriz de datos y nos quedamos con la primera componente principal. Por cada muestra tendremos la primera componente del grupo S y la expresión del gen de interés. Se calcula el coeficiente de correlación de los pares de valores indicados en un grupo y en otro (trabajamos en el caso en que las muestras las clasificamos en dos grupos y se el módulo de la diferencia de los coeficientes de correlación. Obviamente un valor muy grande indica que el coeficiente de correlación en cada uno de los grupos considerados es muy diferente y por tanto el tipo de asociación entre el gen y el grupo de genes es distinto.[219]

[219] Indican que ha implementado el método en el paquete R GPCscore pero no se encuentra en ningún repositorio.

16.14 Un método simple pero efectivo

Es un método propuesto en [69]. Insiste mucho en este trabajo en la innecesaria complejidad del método GSEA comentado en sección § 16.12. Su interés está en la detección de grupos de genes que se asocian con el fenotipo (consideran la situación en que comparamos dos grupos) donde algunos de los genes están sobre expresados (up-regulated) y otros infra expresados (down-regulated).

Como siempre t_i es la medida de asociación marginal gen-fenotipo (puede ser el t-estadístico pero no necesariamente). Si consideramos el grupo de genes S con n_S elementos. Asumiendo (lo que no es cierto) que los distintos t_i son valores observados de variables independientes y con varianza igual a 1, entonces aproximadamente (aplicando teorema central del límite) se tiene que

$$E^{(1)} = \sqrt{n_S}\bar{t}_S \sim N(0,1), \tag{16.11}$$

siendo $\bar{t}_S = \sum_{i\in S} t_i/n_S$. Aunque no hay mayores justificaciones de esta distribución aproximada los autores indican que, sobre los datos, es asumible.[4] Utilizando esta distribución podemos tener fácilmente p-valores que no están basados en la distribución de permutación sino en la distribución normal estándar.

El estadístico considerado en 16.11 detecta cambios en la media pero no cambios de escala. Si un grupo de genes tiene la mitad que están sobre expresados y la otra mitad infra expresados entonces no detectaríamos cambios en la media pero la variabilidad de los valores t_i sí que se modificaría claramente. Proponen utilizar el siguiente estadístico con la siguiente distribución aproximada (con $n_S \geq 20$ proponen los autores)

$$E^{(2)} = \frac{\sum_{i\in S}(t_i - \bar{t}_S)^2 - (n_S - 1)}{\sqrt{2(n_S - 1)}} \sim N(0,1) \tag{16.12}$$

[5] Tendremos un p-valor asociado a cada grupo de genes. Una vez tenemos todos los grupos tenemos un problema de comparaciones múltiples que resolvemos utilizando lo visto en el capítulo § 8. En particular, los autores utilizan los q-valores de Storey. Esencialmente el trabajo

[4] Si la cosa funciona: ¿por qué no?

[5] Bajo la hipótesis de que los t_i son independientes, con una distribución común con varianza unitaria entonces el estadístico tiene una distribución aproximadamente normal estándar. En el trabajo original dividen por $2(n_S - 1)$ y no la raiz cuadrada. Entiendo que debe ser una errata.

se centra mucho en la comparación con el método GSEA destacando su simplicidad frente al otro y que los resultados obtenidos son esencialmente los mismos. Algunos comentarios propios (discutibles):

1. En mi opinión hay un uso de resultados asintóticos que no siempre son asumibles.

2. En cualquier caso, el tamaño de los grupos de genes ha de ser suficientemente grande para que se verifiquen.

3. Es una buena opción a probar.

16.15 Un método basado en poblaciones finitas

Veamos el método propuesto en [101]. Consideremos el conjunto de genes S con m genes.[6] Tenemos t_i la medida de asociación gen-fenotipo para el i-ésimo gen. Consideremos la media muestral como estadístico de enriquecimiento dado por

$$\bar{t}_S = \sum_{i \in S} \frac{t_i}{n_S}.$$

En el resto de aproximaciones lo que se considera aleatorio es la medida de asociación t_i y *fijo* el conjunto de genes S. En este trabajo se adopta otro punto de vista. Tenemos $G = \{1, \ldots, N\}$ un universo de genes y nuestro conjunto de genes S es un *subconjunto aleatorio* de G. Por tanto nos planteamos cuál es la distribución de probabilidad de $\bar{t}_S$ cuando S es un subconjunto aleatorio de tamaño n_S de G o, dicho de otro modo, extraemos al azar y sin reemplazamiento n_S genes del total de N que componen nuestro universo. Los autores llaman a esta distribucion de probabilidad el modelo de conjunto aleatorio (random-set model). Notemos que se plantea la distribución de probabilidad de $\bar{t}_S$ **condicionada a los valores** t_i. Este es el punto básico, los t_i están dados y *no se consideran aleatorios sino fijos, dados previamente.* Utilizando resultados de muestreo en poblaciones finitas se tiene que la media de $\bar{t}_S$ es

$$\mu = E(\bar{t}_S) = \sum_{i \in G} \frac{t_i}{N}, \tag{16.13}$$

es decir, la media de todos los t_i observados sobre todo el universo de genes. La varianza es

$$\sigma^2 = var(\bar{t}_S) = \frac{1}{m} \frac{N-m}{N-1} \left[\sum_{i \in G} \frac{t_i^2}{N} - \left(\sum_{i \in G} \frac{t_i}{N} \right)^2 \right]. \tag{16.14}$$

En este procedimiento no recurrimos a ninguna distribución de aleatorización del estadístico. Estamos utilizando una distribución asintótica en el sentido en que se asume un tamaño de S suficientemente grande para que (aplicando el teorema central del límite) podamos considerar que $\bar{t}_S$ tiene una distribución aproximadamente normal.

[6] Estamos denotando el cardinal de S por n_S. Aquí denotamos por m porque queremos destacar que es un número fijo y dado una vez tenemos el grupo de genes S.

Como estadístico de enriquecimiento utilizan la versión estandarizada de $\bar{t}_S$, es decir, utilizan

$$Z = \frac{\bar{t}_S - \mu}{\sigma} \tag{16.15}$$

con μ y σ dados en las ecuaciones 16.13 y 16.14. Bajo la hipótesis nula de que no hay ningún gen diferencialmente expresado en el conjunto de genes S entonces Z tiene aproximadamente una distribución normal estándar.

$$Z \sim N(0,1). \tag{16.16}$$

16.16 Análisis de grupos de genes por muestra

[220] Single sample gene set analysis

[220]

¿Qué es lo que se pretende en este tipo de procedimientos? Partiendo de una matriz de expresión y una colección de grupos de genes se obtiene una cuantificación **para cada muestra y grupo de genes** de lo diferente que es la expresión en esa muestra del grupo de genes considerado respecto del resto de grupos de genes. De alguna forma lo que estamos atendiendo es a la hipótesis competitiva pero para cada muestra. Pasamos a otra matriz en la que en filas tenemos los grupos de genes considerados mientras que el número de muestras se conserva. Trasladamos el problema de expresión diferencial a nivel de gen[221] a un problema de expresión diferencial a nivel de grupos de genes. Y el valor que asociamos a cada grupo cuantifica lo diferente que es, en la muestra considerada, la expresión de los genes del grupo de la expresión de los genes que no pertenecen a este grupo.

[221] Expresión diferencial marginal en este texto.

Con esta nueva matriz podemos aplicar análisis de expresión diferencial donde en lugar de genes trabajamos con un grupo de genes. De hecho, con una colección de grupos de genes. Esto, obviamente, no tiene ninguna importancia desde el punto de vista estadístico. Podemos aplicar todo lo visto para expresión diferencial marginal.[222]

[222] El adjetivo marginal ahora se refiere a grupos de genes.

16.16.1 GSVA

Este método fue propuesto en [66] y está implementado en el paquete [64, GSVA].

Suponemos $\boldsymbol{x} = [x_{ij}]_{i=1,\ldots,N;j=1,\ldots,N}$ la matriz de expresión obtenida después de normalización previa. Tenemos los grupos de genes $\{S_1 \ldots, S_K\}$ entendidos como subconjuntos de las filas de $\boldsymbol{x}$ ($S_k \subset \{1,\ldots,N\}$. El perfil de expresión del gen (en fila) i será $\boldsymbol{x}_i = (x_{i1},\ldots,x_{in})$. El método trabaja con datos de expresión de microarrays ($\log_2$) o bien con los conteos en datos de RNA-seq.

Estimación kernel de las funciones de distribución Utilizando el perfil de expresión $\boldsymbol{x}_i$ proponen realizar una estimación kernel de la función de distribución. Para datos continuos (microarrays) utilizan

$$\hat{F}_{h_i}(x_{ij}) = \frac{1}{n}\sum_{k=1}^{n} \int_{-\infty}^{\frac{x_{ij}-x_{ik}}{h_i}} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}}\, dt. \tag{16.17}$$

El ancho de banda h_i lo fijan en $h_i = \frac{s_i}{4}$ siendo s_i la desviación estándar de $\boldsymbol{x}_i$.

Para datos de conteo (RNA-seq) utilizan un kernel de Poisson. En concreto el estimador kernel de la función de distribución propuesto es

$$\hat{F}_r(x_{ij}) = \frac{1}{n}\sum_{k=1}^{n}\sum_{y=0}^{x_{ij}} e^{-(x_{ik}+r}\frac{x_{ik}+r}{y!}, \tag{16.18}$$

tomando $r = 0.5$ para hacer que la moda del kernel Poisson coincida con la propia observación. Denotamos por z_{ij} el valor $\hat{F}_{h_i}(x_{ij})$ o $\hat{F}_r(x_{ij})$ dependiendo de que trabajemos con microarrays o con conteos.

Órdenes simétricos Para cada muestra j ordenamos los datos z_{ij} con j constante. Los órdenes (no los valores ordenados) los denotamos por r_{ij}, esto es, $z_{r_1,j} \leq \ldots \leq z_{r_N,j}$. Finalmente los valores r_{ij} son a su vez transformados con el fin de conseguir que sean simétrico alrededor del origen. En concreto tomamos: $\tilde{r}_{ij} = |\frac{N}{2} - r_{ij}|$.

Agregación de puntuaciones por muestra Fijamos j (una muestra dada) y un grupo de genes S_k. y se define el estadístico

$$\nu_{jk}(s) = \frac{\sum_{i=1}^{s}|\tilde{r}_{ij}|^\tau 1_{S_k}(r_{ij})}{\sum_{i=1}^{N}|\tilde{r}_{ij}|^\tau 1_{S_k}(r_{ij})} - \frac{\sum_{i=1}^{s}1_{S_k^c}(r_{ij})}{N-|S_k|}, \tag{16.19}$$

donde $1_{S_k}(r_{ij}) = 1$ si $r_{ij} \in S_k$ y 0 en otro caso. S_k^c es el complementario de S_k y $|S_k|$ el cardinal de S_k.

El valor calculado depende del grupo de genes S_k y de la muestra j.

Resumen En el apartado anterior tenemos una función de s que hemos de resumir. La primera propuesta es tomar

$$E_{jk}^{max} = \nu_{jk}(s*). \tag{16.20}$$

con

$$\nu_{jk}(s*) = \ max\{\nu_{jk}(s) : s = 1, \ldots, N\}. \tag{16.21}$$

Una segunda opción para resumir es considerar

$$E_{jk}^{diff} = \max_{s=1,\ldots,N}\{0, \nu_{jk}(s)\} - \min_{s=1,\ldots,N}\{0, \nu_{jk}(s)\}. \tag{16.22}$$

Los autores proponen los estadísticos en 16.20 y 16.22.

16.17 Otros paquetes y bibliografía complementaria

Un estudio comparativo de distintos métodos de análisis de grupos de genes aparece en [81].

No hemos utilizado en este tema pero son de interés los paquetes [4, topGO] y [46, PGSEA].

La bibliografía sobre este tema es (muy) grande. Como ejemplo, podemos citar como bibliografía complementaria [118, 134, 31, 155, 87, 147, 91, 148, 99, 153, 36, 50].

Una revisión sobre distintos procedimientos para análisis de expresión diferencial basada en conjuntos con datos de RNASeq lo tenemos en [110].

Capítulo 17

Enriquecimiento de grafos

En § 15 y § 16 se ha estudiado la expresión diferencial de grupos de genes en una aproximación que tiene dos pasos. Un primer paso que realiza un análisis de expresión diferencial marginal (o gen a gen). Una vez tenemos este grupo pasamos a un segundo paso en el que enriquecemos la señal que cada uno de los genes nos proporciona. No se ha utilizado el conocimiento previo que podemos tener sobre la interacción de los distintos genes. Por ejemplo, su pertenencia a una determinada ruta de señalización. Los genes se han tratado como un conjunto[223] sin considerar relaciones (interacciones) entre ellos que se suelen modelizar utilizando grafos o redes. De esto va este capítulo. De la incorporación de esta información a la hora de estudiar expresión diferencial quizás no de grupos de genes sino de rutas de señalización o sistemas biológicos en los que estos genes intervienen. En la literatura a este problema se le denomina de distintas formas: gene network enrichment analysis, graph-structured test.

[223] En el sentido matemático de la expresión.

Referencias de interés son [81]

Algunas bases de datos en donde tenemos grafos de interacciones entre genes son:

YeastNet http://www.inetbio.org/yeastnet/[224]

FunCoup http://funcoup.sbc.su.se/.

[224] Probabilistic Functional Gene Network of Saccharomyces cerevisiae.

17.1 Grafos

Un grafo no es más que una colección de vértices y aristas. Siendo un poco más formales el grafo $\mathbb{G}$ sería el par $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ donde $\mathbb{V}$ es el conjunto de vértices mientras que $\mathbb{E}$ es el conjunto de aristas. Podemos denotar $\mathbb{G} = \{1, \ldots, N\}$. Una arista será un par (i, j). Podemos tener grafos dirigidos, no dirigidos o mixtos.

Los vértices del grafo corresponden con genes (proteínas, grupos de genes).

Una arista dirigida indica que el gen en el vértice origen regula al gen en el vértice destino de la arista.

17.2 NEAT

En esta sección consideramos el procedimiento propuesto en [121] e implementado en [122, neat]. Consideramos un grafo cuyos vértices corresponden con genes.

Tenemos un grafo dado describiendo las interacciones entre genes y pretendemos si existe algún tipo de asociación (sobre o infra asociación) entre un par de grupos de genes, A y B. El grupo A podría corresponder a un grupo diferencialmente expresado (obtenido por ejemplo mediante un análisis de expresión diferencial marginal). El conjunto B puede corresponder a un grupo previamente definido en Gene Ontology o en KEGG.

Supongamos en un primer momento un grafo dirigido. Denotamos:

o_A El número de aristas salientes de algún vértice de A.

i_A El número de aristas entrantes a algún vértice de A.

n_{AB} El número de aristas que salen de un vértice de A y llegan a algún vértice de B.

Los autores consideran el siguiente modelo. En ausencia de relación entre los dos grupos de genes tendríamos una urna con i_V bolas (aristas que llegan a algún vértice en el grafo). De ellas consideramos como bolas rojas las que llegan a un vértice de B y negras todas las demás. Por tanto en la urna tenemos i_B bolas rojas e $i_V - i_B$ bolas negras. Extraemos al azar o_A bolas de la urna. Suponiendo que ninguna de estas aristas tienen una mayor probabilidad de llegar a otro vértice que no sea de B entonces el vértice al que llegan podemos considerar que será una extracción aleatoria de la urna considerada. Sea N_{AB} la variable aleatoria que nos da el total de bolas rojas que obtenemos en las o_A bolas extraídas. En otras palabras, N_{AB} cuenta cuántas de las aristas que salen de A llegan a B. Bajo la hipótesis nula de que no hay asociación entre ambos conjuntos de genes este número sería una extracción aleatoria del total de aristas y seguiría una distribución hipergeométrica con o_A extracciones i_B e $i_V - i_B$ bolas negras. Esta sería la distribución nula si no hay asociación de ningún tipo entre A y B. La media de la hipergeométrica es $o_A \frac{i_B}{i_V}$. En [121] se propone contrastar la hipótesis de que la media real de la variable aleatoria N_{AB} es $o_A \frac{i_B}{i_V}$, es decir, contrastar $H_0 : EN_{AB} = o_A \frac{i_B}{i_V}$ frente a $H_1 : EN_{AB} \neq o_A \frac{i_B}{i_V}$. Teniendo en cuenta que N_{AB} tiene una distribución discreta el p-valor sería $p = 2\min\{P(N_{AB} < n_{AB}), P(N_{AB} > n_{AB})\} + P(N_{AB} = n_{AB})$.

Consideremos el caso de un grafo no dirigido. Sea d_A la suma de las aristas que indicen con cualquier vértice de A. Ahora N_{AB} (respectivamente n_{AB}) sería el valor aleatorio (observado) de aristas que unen un vértice de A con algún vértice de B. Bajo la hipótesis nula de no asociación entonces N_{AB} tiene una distribución nula hipérgeométrica con d_A extracciones, d_B bolas rojas y $d_V - d_B$ negras. Por los demás todo es igual.

Cuando tengamos un grafo mixto lo que se hace es las aristas no dirigidas considerarlas como dos aristas dirigidas uniendo los dos vértices en los dos sentidos posibles.

17.3 DEGraph

El procedimiento fue propuesto en [72] y está implementado en el paquete [71, DEGraph].

Método

Se trata de proponer tests multivariantes para estudiar expresión diferencial (básicamente, cambios en el vector de medias) y que sean coherentes con una estructura de grafo dada.

Consideremos un grafo con p genes. Este grafo es $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ donde $\mathbb{V}$ es el conjunto de vértices mientras que $\mathbb{E}$ es el conjunto de aristas. Consideramos el vector $\boldsymbol{\delta} \in \mathbb{R}^p$ que nos indicará la diferencia en la expresión media entre dos condiciones.

Sea $E_{\mathbb{G}}$ una función de energía definida sobre el grafo. ¿Cómo definirla? Veamos distintas posibilidades. Por ejemplo, consideremos la laplaciana del grafo $\mathbb{L}$ Si $\mathbb{G}$ es un grafo no dirigido con matriz de adjacencia A[225] y matriz de grados de incidencia $D = diag(A\mathbf{1}_p)$[226]. El elemento i-ésimo de la diagonal de D, $D_{ii} = d_i$ es el grado de incidencia del vértice i. Se define la matriz laplaciana de $\mathbb{G}$ como $\mathbb{L} = D - A$ y su versión normalizada como $\mathbb{L}_{norm} = I - D^{-1/2}AD^{-1/2}$.

Supongamos que consideramos la función de energía definida sobre un vector $\boldsymbol{\delta} \in \mathbb{R}^d$ como

$$E_{\mathbb{G}}(\boldsymbol{\delta}) = \boldsymbol{\delta}' \mathbb{Q}_{\mathbb{G}} \boldsymbol{\delta}, \tag{17.1}$$

donde $\mathbb{Q}_{\mathbb{G}}$ es $\mathbb{L}$ entonces

$$E_{\mathbb{G}}(\boldsymbol{\delta}) = \sum_{i,j \in \mathbb{V}} (\delta_i - \delta_j)^2 \tag{17.2}$$

mientras que si utilizamos la versión normalizada tendremos

$$E_{\mathbb{G}}(\boldsymbol{\delta}) = \sum_{i,j \in \mathbb{V}} \left(\frac{\delta_i}{\sqrt{d_i}} - \frac{\delta_j}{\sqrt{d_j}} \right)^2 \tag{17.3}$$

De un modo genérico, si consideremos una matriz semidefinida positiva $\mathbb{Q}_{\mathbb{G}}$ entonces podemos considerar su descomposición en valores singulares

$$\mathbb{Q}_{\mathbb{G}} = U \Lambda U', \tag{17.4}$$

siendo U una matriz ortogonal, $\Lambda = diag(\lambda_1, \ldots, \lambda_p)$ los valores propios correspondientes a los vectores propios que forman las columnas de U.

Pretendemos reducir la dimensión del vector $\boldsymbol{\delta}$ a un espacio de dimensión k mucho menor que p de forma que se tengan las funciones de menor energía. De un modo preciso, pretendemos encontrar los vectores $u_i \in \mathbb{R}^p$ con $i \leq k$ tales que

$$u_i = argmin_{v \in \mathbb{R}^p} E_{\mathbb{G}}(v), \tag{17.5}$$

y cada u_i es ortogonal a todos los anteriores u_j con $j < i$. Estos vectores corresponden a los vectores propios asociados a los k valores propios más pequeños.[227]

Se consideran grafos dirigidos $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ donde el conjunto de aristas $\mathbb{E}$ es un par ordenado de vértices. La matriz de adyacencia A

[225] $A = [a_{ij}]_{i,j=1,\ldots,p}$ donde $a_{ij} = 1$ si $(i,j) \in \mathbb{E}$ y cero en otro caso.

[226] $diag(x$ denota la matriz diagonal con el vector x en la diagonal y $\mathbf{1}_p$ es el vector $p \times 1$ con el valor 1 en cada posición.

[227] Vemos que estamos utilizando los mismos resultados que en componentes principales § 12.

puede no ser simétrica y $a_{ij} \neq 0$ si y solamente si $(i,j) \in \mathbb{E}$ o, de otro modo, una arista va del vértice v_i al vértice v_j. Se utiliza la función de energía

$$E_{\mathbb{G}}(\boldsymbol{\delta}) = \sum_{i:d_i^- \neq 0}^{p} \left(\delta_i - \frac{1}{d_i^-} \sum_{(j,i)\in\mathbb{E}} a_{ij}\delta_j \right)^2, \tag{17.6}$$

donde $d_i^- = \sum_{j=1}^{p} |a_{ji}|$. Se comprueba que

$$E_{\mathbb{G}}(\boldsymbol{\delta}) = \boldsymbol{\delta}' M_{\mathbb{G}} \boldsymbol{\delta}, \tag{17.7}$$

siendo $M_{\mathbb{G}} = (\tilde{I} - D_-^{-1}A')'(\tilde{I} - D_-^{-1}A')$, $D_- = diag(d_1^-, \ldots, d_p^-)$ y $\tilde{I} = diag(I(d_1^- \neq 0), \ldots, I(d_p^- \neq 0))$[228] Una vez tenemos esta forma cuadrática podemos hacer su descomposición en valores singulares, $M_{\mathbb{G}} = U'\Lambda U$. En lo que sigue si $x \in \mathbb{R}^p$ entonces se denota $\tilde{x} = U'x$.

[228] El elemento $\tilde{I}$ es no nulo si $d_i^- \neq 0$ y nulo en otro caso.

El procedimiento propuesto consiste en lo siguiente: en lugar de trabajar con los datos originales x se trabaja con las primeras k columnas de la matriz U correspondientes a los k vectores propios cuyos valores propios son los k **menores**. Y se aplica sobre estos datos reducidos un test de la T^2 de Hotelling.[229] Varios comentarios son importantes. Obviamente no podemos aplicar el test de la T^2 de Hotelling por un problema de dimensión. Los autores muestran además la ganancia de potencia en el test de comparación aplicando esta reducción de dimensión. ¿Cómo queda el procedimiento? Denotamos:

[229] https://en.wikipedia.org/wiki/Hotelling's_T-squared_distribution.

1. $\bar{x}_i$ con $i = 1, 2$ los vectores de medias en ambas condiciones.
2. S la matriz de covarianzas estimada conjuntamente[230]

[230] Pooled estimator.

Si aplicáramos el test de T^2 de Hotelling entonces el estadístico del contraste sería

$$T^2 = \frac{n_1 n_2}{n_1 + n_2}(\bar{x}_1 - \bar{x}_2)' S^{-1} (\bar{x}_1 - \bar{x}_2). \tag{17.8}$$

Si en lugar de trabajar en la base original transformamos a una base ortonormal formada por las columnas de U entonces el estadístico no cambiaría. Si denotamos el estadístico en la nueva base $\tilde{T}^2$ se tiene que

$$T^2 = \tilde{T}^2 = \frac{n_1 n_2}{n_1 + n_2}(\bar{x}_1 - \bar{x}_2)' U (U'SU)^{-1} U' (\bar{x}_1 - \bar{x}_2). \tag{17.9}$$

Realmente no se trabaja con la nueva base completa sino con k componentes ($k \leq p$). Denotamos por $U_{[k]}$ la matriz formada por las primeras k componentes de U (correspondientes a los k menores valores propios) y por $I_{[k]}$ una matriz con todo ceros salvo los k primeros elementos de la diagonal principal iguales a uno. Considerando la proyección sobre el espacio generado por las primeras k componentes el estadístico de Hotelling en el nuevo espacio sería

$$\begin{aligned} \tilde{T}_k^2 = &\frac{n_1 n_2}{n_1 + n_2}(\bar{x}_1 - \bar{x}_2)' U_{[k]} (U_{[k]}' S U_{[k]})^{-1} U_{[k]}' (\bar{x}_1 - \bar{x}_2) = \\ &\frac{n_1 n_2}{n_1 + n_2}(\bar{x}_1 - \bar{x}_2)' U I_{[k]} U' (U I_{[k]} U' S U I_{[k]} U')^{+} U I_{[k]} U' (\bar{x}_1 - \bar{x}_2). \end{aligned} \tag{17.10}$$

donde A^+ es la inversa generalizada de la matriz A. El estadístico dado en 17.10 es el utilizado en el contraste.

Ejemplo

```
library(DEGraph)
```

Parte VI

Metaanálisis

Capítulo 18

Metaanalisis

[1]

18.1 Bibliografía

Metaanalisis tiene una muy extensa bibliografía. Y con niveles estadísticos muy diferentes. Creo conveniente hacer algunos comentarios previos.

1. [119] y [107] son textos con un buen nivel estadístico.
2. Para el uso este tipo de técnicas con R podemos consultar [38] y https://bookdown.org/MathiasHarrer/Doing_Meta_Analysis_in_R/#ref-pigott2012advances

18.2 Introducción

La bibliografía crece. Y hay que intentar obtener conclusiones de esa masa de publicaciones. Obtener algún tipo de conclusión a partir de diferentes estudios. De eso nos ocupamos.

[1] ¿Metaanálisis, metanálisis o meta-análisis?
Voy a contestar a la manera de mi suegro cuando le hice una mera pregunta de cortesía sobre cierto tema político actual: "Bueenoooo, pues esto debemos remontarnos a Páez y a Bolívar cuando...". Pues eso, toma asiento, que para eso "catedrático" es el que se sienta en la silla. Los ordenadores no tienen nunca dudas porque su lenguaje es unívoco y sus reglas fijas. O sea, no hablan. Pero las lenguas vivas, como los hombres, están llenas de contradicciones porque son el resultado de las diversas circunstancias históricas. El prefijo "meta" (más allá) es griego y solamente los de la casa y algunos invitados sabemos que "metáfora" y "metafísica" son palabras compuestas. La forma distintiva "meta-análisis", por desgaste, terminará en "metaanálisis" y, dando una vuelta de tuerca, en "metanálisis", simplificando las dos vocales. Hoy ya se puede escribir "contraataque" y "contrataque". Las dos formas son válidas para la RAE ("AE" cuando venga la III Repú.). Por consiguiente - Felipe dixit - yo diría "metaanálisis" (más aún siendo vuestro ámbito verbal tan reducido que nadie os lee salvo vosotros mismos). Pero esto no es un dogma sino una opinión.
Por lo demás, aunque la Academia no ponga multas y tenga cierta autoridad moral sobre el idioma, tampoco hay que llevarle el café a la mesa como si fuésemos meritorios. Hoy permite decir "sicología" y yo te juro por mis ocho apellidos aragoneses que no me arrancarán la "p" aunque una docenas de académicos me manteen como a Sancho Panza los burlones arrieros. En suma, en materia de lenguaje, lo que el hombre ha unido el hombre lo puede separar. Pablo Galindo Arlés 9/12/2019.

Tenemos una hipótesis que queremos evaluar. No tenemos datos propios o no queremos hacer **nuestro** experimento solamente. Queremos evaluar qué dice la literatura sobre la pregunta que nos hacemos. El primer paso (y en ocasiones último) es hacer una búsqueda no sesgada de la bibliografía. A esto se le denomina hacer una **revisión sistemática**

[**Los paquetes de Bioconductor para realizar meta-analisis son los siguientes \cite {Gene metaArray**, **RankProd}**]

18.3 Combinando p-valores

18.4 Listas ordenadas de genes

[2] Cuando se realizan distintos estudios de expresión diferencial lo que básicamente obtenemos son listas de genes que están ordenados [3] Estas listas ordenadas pueden proceder de un mismo estudio en donde aplicamos distintos procedimientos. Podemos tener distintos experimentos analizados del mismo modo y que luego pretendemos comparar a partir de la lista ordenada que hemos observada. ¿Cómo de coincidentes son las listas? Una revisión muy interesante es [19].

En [104] se propone un método y se realiza un estudio con microarrays. Se muestra la fuerte dependencia de la plataforma utilizada.

En lo que sigue seguimos [19] utilizando el paquete desarrollado por los autores [**GeneSelector**].

18.5 Otros procedimientos

Una posibilidad es tomar datos obtenidos con distintas plataformas. Se realiza un análisis cluster y se comprueba que las clasificaciones de las muestras que obtenemos corresponden a criterios biológicos y no técnicos. Esto es, que los grupos de muestras que observamos se refieren a tipo de cáncer o tipo de tejido y no al plataforma utilizada. Un ejemplo es [**Jong07**] en donde se utilizan arrays CGH.

18.6 Otra bibliografía de interés

El trabajo [47] es una herramienta recientemente publicada para hacer meta-análisis utilizando datos Affymetrix. Es una herramienta basada en el uso de Kepler que realiza de un modo automatizado todo el proceso que necesitamos para realizar un meta-análisis incluyendo bajar los datos, procesarlos, análizarlos individualmente y compararlos. Utiliza además de R otros programas.

[2] Esta sección está basada en el artículo [19] y en el paquete [**GeneSelector**].

[3] También podemos pensar en ordenar grupos de genes de acuerdo a algún criterio de análisis de grupo de genes o análisis de enriquecimiento.

18.7 Meta-análisis con microarrays

18.8 Bibliografía comentada

[104] es un estudio en donde se comparan distintos estudios de expresión diferencial y de coexpresión. El interés está en cómo comparan las listas ordenadas.

[**GeneSelector**] es un paquete R/Bioconductor que implementa muchos métodos para la ordenación de listas de genes. Un review de los métodos utilizados es [19].

Capítulo 19

Listas de características ordenadas

[231]

En temas anteriores hemos visto cómo cuando realizamos un análisis de cada característica marginalmente obtenemos un estadístico y un p-valor asociado. Por ejemplo, en la situación más simple de comparación de dos grupos de muestras (dos condiciones) podemos utilizar un t-test (bilateral). De hecho ordenamos las características según el módulo del estadístico es mayor o bien si el p-valor es más pequeño. Ambos criterios producen la misma ordenación.[232]

En lo anterior estas listas las hemos usado sólamente a fin de determinar en qué posición cortar de modo que por encima del punto de corte declaramos que un test es significativo y por debajo que no lo es. ¿Este es el único uso que podemos realizar de esta ordenación? No, claro. Supongamos que obtenemos 1000 características significativas con un FDR de 0.05. ¿De qué nos sirve? ¿El investigador va a seguir estudiando cada uno de estos genes? No es probable que tenga dinero para ello.[233] Lo más probable es que elija seguir estudiando aquellos genes que aparecen en la parte superior de la lista que hemos construido. Por tanto, la posición en la lista tiene gran importancia.

¿Cómo de estable es la ordenación que hemos obtenido? Cuando se habla de estable hay que definir frente a qué se pretende estabilidad. Una primera fuente de variabilidad a la hora de obtener la lista es utilizar distintos estadísticos (y por lo tanto distintos p-valores) cuando contrastamos la expresión diferencial marginal. Por tanto, estabilidad frente al procedimiento estadístico utilizado. Dentro de este apartado se puede incluir el hecho que los datos hay que preprocesarlos (correcciones de fondo, normalización y resumen en microarrays por ejemplo) y no hay, por supuesto, un único método.

Una segunda fuente de variabilidad son modificaciones en los datos que manejamos, pequeñas modificaciones de los mismos no debieran afectar (demasiado) a la lista obtenida.

En consecuencia, la lista ordenada es variable y hemos de cuantificar la estabilidad de la ordenación. Esto se trata en § 19.1. Un segundo aspecto es cómo agregar distintas listas con objeto de obtener una lista agregada más estable, más fiable. Lo vemos en § 19.2.

[231] Este capítulo está basado fundamentalmente en [19] y su implementación en el paquete [124, GeneSelector]. De hecho, no es más que un resumen. Una buena referencia es [32].

[232] Si es con el estadístico ordenamos de mayor a menor y si es con el p-valor la ordenación se hace de menor a mayor.

[233] Bueno, a lo mejor en Estados Unidos o Alemania. Eso dicen.

19.1 Estabilidad de la lista ordenada

Tenemos la matriz de expresión $\boldsymbol{x} = [x_{ij}]_{i=1,\ldots,N;j=1,\ldots,n}$ y el vector que describe las muestras $\boldsymbol{y} = (y_1, \ldots, y_n)'$. Una lista ordenada es una permutación de $(1, \ldots, N)$ que podemos denotar $\pi(i)$. El gen i ocupa la posición $\pi(i)$ en la lista ordenada. Podemos también denotar por $(r_1, \ldots, r_N)$ la lista ordenada de modo que

$$r_j = i \iff \pi(i) = j.$$

Por ejemplo, si $\pi(1898) = 1$ significa que el gen cuyo perfil de expresión está en la fila 1989 está en la primera posición de la lista ordenada. Tendremos también que $r_1 = 1989$. No vamos a considerar ordenaciones con empates. Normalmente se suele mostrar la lista de los k primeros genes en la lista[234] donde k suele ser 20 o 50.

[234] Top-k list.

19.1.1 Diferentes criterios de ordenación

Si nos centramos en el caso de dos grupos tenemos muchísimos procedimientos que nos permiten cuantificar la asociación entre la expresión del gen y el fenotipo.

Cargamos el paquete [124, GeneSelector].

```
library(GeneSelector)
```

Utilizamos los datos tamidata::gse1397.

```
library(tamidata)
data(gse1397)
x = exprs(gse1397)
y = pData(gse1397)[,"type"]
```

Obtengamos distintos estadísticos. Empezamos con el fold-change

```
fc = RankingFC(x,y)
```

Calculamos los t-estadísticos.

```
tstat = RankingTstat(x,y)
```

Los t-estadísticos moderados utilizando el modelo limma [125]

```
limma = RankingLimma(x,y,proportion=0.01)
```

Los t-estadísticos de Fox-Dimmic [**FoxDimmic06**].

```
foxdimmic = RankingFoxDimmic(x,y,m=100)
```

El t-estadístico de contracción[235] con

[235] Shrinkage t-statistic.

```
shrinkt = RankingShrinkageT(x,y)
```

Nos fijamos para comentar en detalle en una de estas ordenaciones. Veamos qué nos han devuelto.

```
class(tstat)

## [1] "GeneRanking"
## attr(,"package")
## [1] "GeneSelector"
```

Que tiene los siguientes slots.

```
getSlots("GeneRanking")

##           x           y   statistic     ranking
##    "matrix"    "factor"   "numeric"   "numeric"
##        pval        type      method
##    "vector" "character" "character"
```

Por ejemplo, en `tstat@x` tenemos la matriz de expresión y `tstat@y` nos proporciona la variable fenotípica que en este caso simplemente nos da el grupo al que pertenece la muestra (en código numérico.

```
tstat@y

##  [1] 1 1 1 1 1 1 1 2 2 2 2 2 2 2
## Levels: 1 2
```

La lista la tenemos en `tstat@ranking`. Veamos los primeros.

```
head(tstat@ranking)

## 1007_s_at   1053_at    117_at    121_at 1255_g_at
##     12194     12433      2870      8757     15880
##   1294_at
##       234
```

Nos indica la posición en la matriz de expresión y también tiene como etiqueta el identificador Affy (AffyID). También podemos ver los t-estadísticos y los correspondientes p-valores

```
head(tstat@statistic)

##  1007_s_at    1053_at     117_at     121_at
## -0.6183648  0.6022344  1.5635785 -0.8738516
##  1255_g_at    1294_at
##  0.3815208  2.9096918

head(tstat@pval)

##  1007_s_at    1053_at     117_at     121_at
## 0.54789263 0.55822637 0.14389203 0.39934953
##  1255_g_at    1294_at
## 0.70948639 0.01308856
```

y recuperar la información sobre el test que hemos utilizado que en este caso es un t-test para muestras independientes.

```
tstat@type

## [1] "unpaired"

tstat@method

## [1] "ordinaryT"
```

La función GeneSelector::toplist nos proporciona un resumen útil con la lista en la primera columna, el estadístico en la segunda y el p-valor en la tercera. Podemos ver las 10 primeras posiciones con

```
toplist(tstat,top=10)

##    index  statistic         pval
## 1    346 -10.517552 2.070655e-07
## 2   1170 -10.344123 2.481752e-07
## 3   1853  -8.762265 1.463473e-06
## 4   6303   7.648232 5.939862e-06
## 5   7889  -7.190209 1.101446e-05
## 6  10450   6.365838 3.579137e-05
## 7  10132  -6.020134 6.027785e-05
## 8  17751  -6.013987 6.084819e-05
## 9  16324  -5.798600 8.491862e-05
## 10 12401   5.637777 1.093774e-04
```

Hasta ahora hemos descrito los resultados utilizando el t-test. Hemos utilizado otros tres. Mostremos de un modo conjunto los resultados.

```
fc.top = toplist(fc,top=10,show=FALSE)
tstat.top = toplist(tstat,top=10,show=FALSE)
limma.top = toplist(limma,top=10,show=FALSE)
foxdimmic.top = toplist(foxdimmic,top=10,show=FALSE)
shrinkt.top = toplist(shrinkt,top=10,show=FALSE)

global.top = cbind(fc.top[,"index"],tstat.top[,"index"],limma.top[,"index"],
      foxdimmic.top[,"index"],shrinkt.top[,"index"])
colnames(global.top) = c("fc","tstat","limma","foxdimmic","shrinkt")
```

Y obtenemos.

```
global.top

##          fc tstat limma foxdimmic shrinkt
##  [1,]  9149   346   346      6303     346
##  [2,] 11148  1170  1170      6468    1170
##  [3,]  4375  1853  1853       328    1853
##  [4,]  8951  6303  6303       651    6303
##  [5,]   888  7889  7889      8566   10132
##  [6,]  3946 10450 10132      9149   17751
##  [7,] 11145 10132 17751     11148   16324
##  [8,] 11150 17751 16324      9620   14969
##  [9,] 11573 16324 14969      5817     650
## [10,] 11659 12401  9555       966    2277
```

¿Tenemos genes que aparecen en las cinco listas consideradas? Podemos observar que no hay ninguno. La función GeneSelector::GeneSelector nos hace el trabajo de comprobar si hay genes presentes en la parte superior según todos los criterios.

```
lista = list(fc,tstat,limma,foxdimmic,shrinkt)
genesel = GeneSelector(lista,maxrank=10)
show(genesel)

## GeneSelector run with gene rankings from the following statistics:
## Foldchange
## ordinaryT
## Limma
## FoxDimmicT
## ShrinkageT
## Number of genes below threshold rank 10 in all statistics:0
```

Sin embargo, excluyendo el primer criterio (que no es precisamente el mejor) obtenemos un resultado distinto.

```
listareducida = list(tstat,limma,foxdimmic,shrinkt)
genesel = GeneSelector(listareducida,maxrank=10)
show(genesel)

## GeneSelector run with gene rankings from the following statistics:
## ordinaryT
## Limma
## FoxDimmicT
## ShrinkageT
## Number of genes below threshold rank 10 in all statistics:1
```

Podemos ver qué gen es.

```
which(slot(genesel,"selected") == 1)

## [1] 6303
```

Repitiendo para los 50 primeros de la lista reducida.

```
genesel = GeneSelector(listareducida,maxrank=50)
show(genesel)

## GeneSelector run with gene rankings from the following statistics:
## ordinaryT
## Limma
## FoxDimmicT
## ShrinkageT
## Number of genes below threshold rank 50 in all statistics:2
```

Y son los siguientes.

```
which(slot(genesel,"selected") == 1)

## [1]  651 6303
```

Y una tentación a evitar. Cuando un investigador ha invertido tiempo y dinero (y necesita publicar desesperadamente) necesita obtener resultados "significativos". Posiblemente tenía la hipótesis de que algún o algunos genes debieran de estar en la parte superior de la lista. La cosa es fácil. ¿Por qué no elegir estadísticos que lo hagan entre los disponibles o incluso proponiendo uno nuevo? En definitiva elegir los criterios utilizando información previa (a veces, sin mala intención). Obviamente esto sesga los resultados o simplemente los invalida.[236]

[236] Y el comentario es válido para muchos otros procedimientos estadísticos. Siempre hay un evidente sesgo hacia la publicación de los resultados significativos ignorando los que no lo son.

19.1.2 Estabilidad frente a cambios en los datos

¿Qué ocurre con nuestra lista si quitamos alguna muestra?[237] ¿O si cambian un poco los valores? En definitiva, esperamos que pequeñas modificaciones en los datos no alteren en demasía nuestra ordenación de las características.

[237] De las pocas que habitualmente tendremos.

Veamos diferentes opciones para modificar los datos que lleva implementados [124, GeneSelector].

La primera opción puede ser elegir un número de muestras k ($< n$ el número total de muestras) y *quitarlas* de nuestros datos. El número total de posibilidades será el número combinatorio $\binom{n}{k}$.[238] Este procedimiento es conocido como *Jackknife* . Un caso particular sería cuando $k = 1$ y vamos dejando una muestra fuera cada vez. Sería el procedimiento *leaving-one-out* . Cuando aplicamos este procedimiento podemos considerar las $\binom{n}{k}$ o bien una muestra elegida al azar si son muchas.

[238] Que nos da el número de subconjuntos distintos de tamaño k que podemos formar con un total de n elementos.

Una segunda opción es utilizar un método bootstrap. En este caso lo que hacemos es elegir de las n muestras disponibles un total de n muestras con reeemplazamiento. Obviamente alguna de las muestras se puede repetir.[239]

[239] El método bootstrap consiste en muestrear sobre los propios datos. O dicho de un modo más técnico muestrear de la distribución empírica de las muestras.

Una tercera opción puede ser añadir ruido aleatorio a los propios datos de expresión. En microarrays podemos añadir ruido normal con media nula y un valor pequeño de la varianza que tiene que tener en cuenta el orden de los datos.[240]

[240] Lo que se conoce como ruido blanco.

Una cuarta opción es modificar la covariable y. Supongamos que tenemos un factor experimental (un número finito de grupos) y lo que hacemos es cambiar la etiqueta que identifica el grupo sobre una pequeña cantidad de muestras. En resumen cambiamos el grupo de un pequeño grupo de muestras.

Fijamos la semilla del generador para obtener los mismos resultados.

```
set.seed(1979)
```

Aplicamos jackknife con dejando fuera cada vez una de las muestras originales. [241] Observemos que se impone una restricción a esto y es que cada clase ha de tener un tamaño de, al menos, 6 muestras.

[241] En definitiva, un leaving-one-out.

```
leave1out = GenerateFoldMatrix(y = as.numeric(y), replicates = 50,
    k = 1,minclassize=6)
show(leave1out)

## number of removed samples per replicate: 1
## number of replicates: 14
## constraints: minimum classize for each class: 6
```

Y la salida la podemos pasar a la función GeneSelector::RepeatRanking.

```
leave1out.tstat = RepeatRanking(tstat,leave1out,scheme="subsampling")
```

Un dibujo de interés es comparar para cada gen su posición original y las que tiene con los nuevos datos perturbados o modificados. Es la figura 19.1.

```
png(paste0(dirTamiFigures,"ListasOrdenadas25.png"))
plot(leave1out.tstat)
dev.off()
```

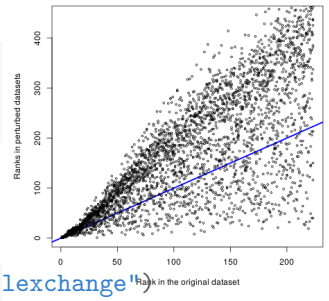


Figura 19.1

También podemos modificar los datos de modo que vamos cambiando la etiqueta de una muestra cada vez.[242]

```
change1.tstat = RepeatRanking(tstat,leave1out,scheme="labelexchange")
```

[242] Lo que significa que asignamos esa muestra al otro grupo.

También generamos muestras bootstrap imponiendo la misma restricción sobre el tamaño de cada clase.

```
boot = GenerateBootMatrix(y = as.numeric(y), replicates = 50,
    maxties = 3,minclassize=6)
boot.tstat = RepeatRanking(tstat,boot)
```

Podemos añadir ruido a las intensidades observadas.

```
noise.tstat = RepeatRanking(tstat, varlist=list(genewise=TRUE, factor=1/10))
```

Fijémonos ahora a los resultados obtenidos con las muestras generadas mediante bootstrap.[243]

[243] Resultados no mostrados.

```
toplist(boot.tstat,show = FALSE)
```

Vemos que nos muestra la tabla original con los diez primeros genes, el t-estadístico y el p-valor. La siguiente tabla de frecuencias muestras los genes que han aparecido al menos una vez entre los 10 primeros y la frecuencia de veces que aparece en cada una de las 10 primeras posiciones.

En la figura 19.2 mostramos las ordenaciones originales en abscisas frente a las observadas cuando modificamos los datos según los distintos procedimientos. Es más que evidente la tremenda variabilidad que obtenemos en las ordenaciones.

```
par(mfrow=c(2,2))
plot(leave1out.tstat, col="blue",
 pch=".", cex=2.5, main = "jackknife")
plot(change1.tstat, col="blue",
 pch=".", cex=2.5, main = "intercambio etiqueta")
plot(boot.tstat, col="blue",
 pch=".", cex=2.5, main = "bootstrap")
plot(noise.tstat, frac=1/10,
 col="blue", pch=".", cex=2.5, main = "ruido")
```

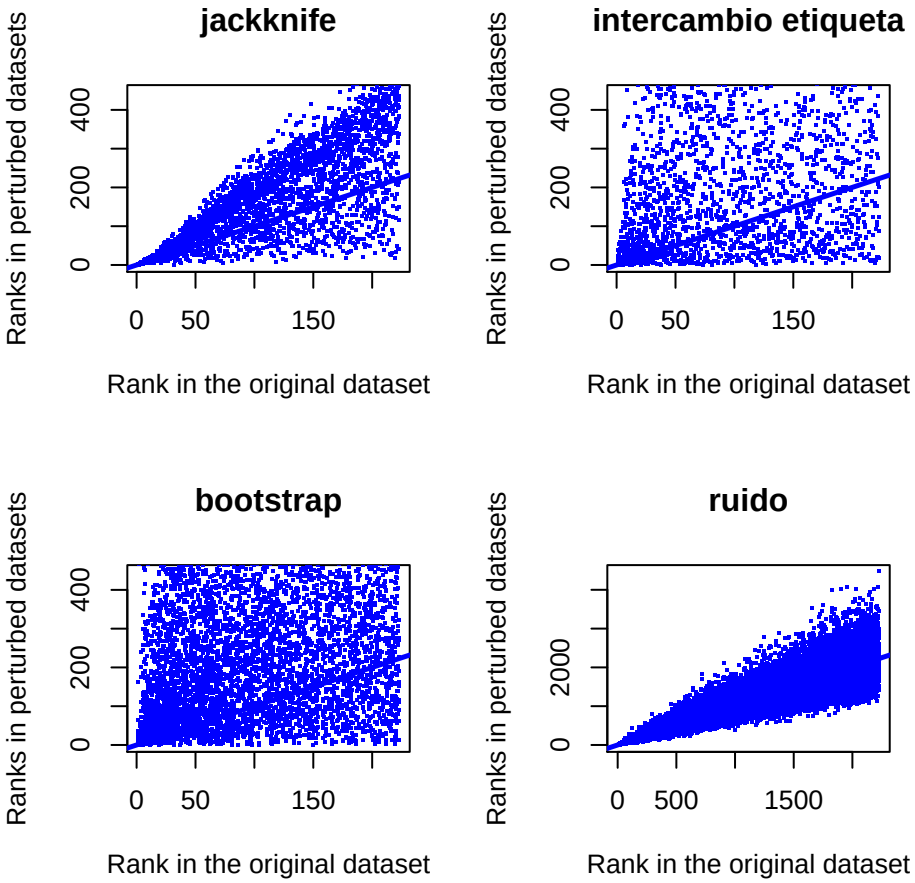


Figura 19.2: Orden original frente a los obtenidos modificando los datos.

19.2 Agregación de listas

Hemos obtenido cinco listas distintas correspondientes a distintos estadísticos. Vamos a plantearnos la obtención de una sola lista que las agregue. Lo primero[244] es construir un heatmap. Los datos a clasificar son las distintas ordenaciones. Se realiza una doble clasificación jerárquica con la distancias euclídea[245] y el enlace completo como disimilaridad entre grupos.[246]

En la figura 19.3 tenemos el dibujo. Se aprecia cómo el criterio del fold change produce una lista claramente distinta a las demás. También es claro que el t-estadístico y el t-estadístico modera de Limma producen ordenaciones similares. En cuanto a los genes se necesita un estudio mucho más profundo para concluir algo.[247]

[244] Porque es bonito aunque tengo mis serias dudas de que sirva para mucho.

[245] Notemos que son órdenes y no está demasiada justificada esta elección.

[246] § 13.

[247] El dibujo es bonito y por eso lo he puesto.

```
merged = MergeMethods(lista)
```

```
HeatmapRankings(merged)
```

```
merged = MergeMethods(lista)
HeatmapRankings(merged)
```

Figura 19.3: Heatmap para las ordenaciones obtenidas con distintos estadísticos.

Hemos de agregar las ordenaciones. La opción más básica sería promediar, por gen, los órdenes obtenidos.

```
AggMean = AggregateSimple(merged, measure = "mean")
```

Parte VII

Algo de Estadística

Capítulo 20

Conceptos fundamentales de Estadística

El contenido tratado en los temas anteriores es intencionadamente muy básico. Se pretende llegar al lector que no tiene ningún conocimiento de Probabilidad y Estadística. Sin embargo, hay procedimientos que se tratan en este manual y que requieren un nivel de formalización mayor. En este tema se aborda este otro nivel. Para una primera lectura del manual no se requiere. Pero sí para una lectura completa. El lenguaje utilizado es más preciso y directo.

20.1 Verosimilitud

Sea $y = (y_1, \ldots, y_n)$ una realización del vector aleatorio $Y = (Y_1, \ldots, Y_n)$. Es habitual asumir que Y tiene una función de densidad conjunta f en una cierta familia $\mathcal{F}$. Para una función dada f, el valor $f(y)$ nos muestra cómo varía la densidad dentro del espacio muestral de valores posibles de y. Y viceversa, si consideramos unos datos y y lo que hacemos variar es la función de densidad entonces estamos viendo cómo de verosímil es cada una de las funciones dados los datos y. Esta función recibe el nombre de **verosimilitud** de f dados los datos y y se suele denotar como

$$Verosimilitud[f; y] = L(f; y) = f(y). \tag{20.1}$$

Con frecuencia, es conveniente trabajar con el logaritmo natural de la función anterior y hablaremos de la **log-verosimilitud**.

$$l[f; y] = \log f(y). \tag{20.2}$$

Una simplificación adicional (que es habitual en las aplicaciones) supone que la función de densidad f pertenece a una familia paramétrica $\mathcal{F}$, esto es, cada elemento de la familia es conocido completamente salvo un número finito de parámetros $\theta = (\theta_1, \ldots, \theta_p)$ de modo que denotaremos $f(y; \theta)$ o $f_Y(y; \theta)$. Al conjunto de valores posibles de θ se le llama **espacio paramétrico** y lo denotaremos por Θ. En este caso, la logverosimilitud es una función de θ y denotaremos

$$Verosimilitud[\theta; y] = l(\theta; y) = \log f(y; \theta). \tag{20.3}$$

Supongamos una transformación 1-1 de Y a Z, $Z = g(Y)$. Las densidades de ambos vectores se relacionan según la siguiente relación

$$f_Z(z) = f_Y(y)\left|\frac{\partial y}{\partial z}\right|,$$

donde $\left|\frac{\partial y}{\partial z}\right|$ es el jacobiano de la transformación de z a y. Se tiene la siguiente relación entre las verosimilitudes

$$L_Z(\theta; z) = \left|\frac{\partial y}{\partial z}\right| L_Y(\theta; y).$$

Esto sugiere que es mejor trabajar con el cociente de las verosimilitudes para dos vectores de parámetros θ_1 y θ_2 en lugar de los valores aislados.

Si asumimos que los distintos $Y_1, \ldots, Y_n$ son independientes entonces

$$L_Y(\theta; y) = f_Y(y) = \prod_{i=1}^{n} f_{Y_i}(y_i),$$

y

$$l_y(\theta; y) = \sum_{i=1}^{n} \log f_{Y_i}(y_i) = \sum_{i=1}^{n} L_{Y_i}(\theta; y_i).$$

Veamos algunos ejemplos de verosimilitud.

Ejemplo 20.1 (Pruebas Bernoulli) *$Y_1, \ldots, Y_n$ son independientes y con la misma distribución (i.i.d.) $P(Y_i = y_i) = \theta^{y_i}(1-\theta)^{1-y_i}$ y*

$$L(\theta; y) = \theta^{\sum_{i=1}^{n} y_i}(1-\theta)^{n-\sum_{i=1}^{n} y_i}$$

Ejemplo 20.2 (Número de éxitos en n pruebas Bernoulli) *Nuestros datos son ahora el número total de éxitos en un número dado de pruebas de Bernoulli, r. Entonces la variable correspondiente R tiene una distribución binomial con n pruebas y una probabilidad de éxito θ. La verosimilitud viene dada por*

$$L(\theta; r) = \binom{n}{r}\theta^r(1-\theta)^{n-r}$$

Ejemplo 20.3 (Muestreo Bernoulli inverso) *Nuestros datos son ahora el número total de pruebas necesarias para alcanzar un número previamente especificado de éxitos. La variable aleatoria correspondiente N tendrá una distribución binomial negativa con r éxitos y una probabilidad de éxito θ. La función de verosimilitud correspondiente viene dada por*

$$L(\theta; n) = \binom{n-1}{r-1}\theta^r(1-\theta)^{n-r}$$

Consideremos los tres ejemplos anteriores 20.1, 20.2 y 20.3. Si consideramos dos valores del parámetro θ_1 y θ_2 entonces el cociente de las verosimilitudes calculados en ambos valores tiene el mismo valor en los tres ejemplos.

20.2 Estimación

Denotamos por Θ el espacio formado por los valores que puede tomar θ o espacio paramétrico. Un **estimador** del parámetros o vector paramétrico θ es cualquier función de la muestra $X_1, \ldots, X_n$ que toma valores en el espacio paramétrico. Si $\delta(X_1, \ldots, X_n)$ es un estimador del parámetro θ entonces se define el **error cuadrático medio** como

$$MSE(\delta) = E[\delta(X_1, \ldots, X_n) - \theta]^2 \tag{20.4}$$

En el caso en que se verifique que $E\delta(X_1, \ldots, X_n) = \mu_\delta = \theta$, es decir, que el estimador sea **insesgado** entonces:

$$MSE(\delta) = E[\delta(X_1, \ldots, X_n) - \theta]^2 = E[\delta(X_1, \ldots, X_n) - \mu_\delta]]^2 = var(\delta).$$

Y el error cuadrático medio no es más que la varianza del estimador. Consideremos la siguiente cadena de igualdades. Denotamos

$$MSE(\delta) = E[\delta - \theta]^2 = E[\delta - \mu_\delta + \mu_\delta - \theta]^2 = E[\delta - \mu_\delta]^2 + [\mu_\delta - \theta]^2 \tag{20.5}$$

La diferencia entre la media del estimador y el parámetro, $\mu_\delta - \theta$, recibe el nombre de **sesgo**. Finalmente lo que nos dice la ecuación anterior es que el error cuadrático medio $MSE(\delta)$ lo podemos expresar como la suma de la varianza del estimador, $E[\delta - \mu_\delta]^2$, más el sesgo al cuadrado, $[\mu_\delta - \theta]^2$.

A la raíz cuadrada de la varianza de un estimador, es decir, a su desviación típica o estándar se le llama **error estándar**. La expresión error estándar se usa en ocasiones indistintamente para referirse o bien dicha desviación típica o bien al estimador de la misma.

20.2.1 Estimación insesgada de media y varianza

Dada una muestra $Y_1, \ldots, Y_n$ de una variable. Un estimador habitualmente utilizado para estimar $\mu = EY_i$ es la media muestral dada por

$$\bar{Y} = \frac{1}{n} \sum_{i=1}^{n} Y_i. \tag{20.6}$$

Notemos que

$$E\bar{Y} = E[\frac{1}{n} \sum_{i=1}^{n} Y_i] = \frac{1}{n} \sum_{i=1}^{n} EY_i = \frac{1}{n} \sum_{i=1}^{n} \mu = \mu.$$

En definitiva, la media muestral es un estimador que no tiene ningún sesgo cuando estima la media de Y_i (la media poblacional) o, lo que es lo mismo, es un estimador **insesgado**.

Para estimar de un modo insesgado la varianza σ^2 a partir de una muestra $Y_1, \ldots, Y_n$ se utiliza la varianza muestral dada por

$$S^2 = \frac{1}{n-1} \sum_{i=1}^{n} (Y_i - \bar{Y})^2. \tag{20.7}$$

La razón de la división por $n - 1$ en lugar de dividir por n viene de

las siguientes igualdades.

$$E\sum_{i=1}^{n}(Y_i-\bar{Y})^2 =$$
$$E\sum_{i=1}^{n}[(Y_i-\mu)-(\bar{Y})^2-\mu)]^2 = \sum_{i=1}^{n}E(Y_i-\mu)^2 - nE(\bar{Y}-\mu)^2, \tag{20.8}$$

pero $E(Y_i-\mu)^2=\sigma^2$ y $E(\bar{Y}-\mu)^2 = var(\bar{Y}) = \sigma^2/n$. En consecuencia,

$$E\sum_{i=1}^{n}(Y_i-\bar{Y})^2 = n\sigma^2 - \frac{\sigma^2}{n} = n\sigma^2-\sigma^2 = (n-1)\sigma^2,$$

de donde,

$$ES^2 = E\frac{1}{n-1}\sum_{i=1}^{n}(Y_i-\bar{Y})^2 = \sigma^2,$$

es decir, S^2 estima la varianza σ^2 sin sesgo.

20.2.2 Estimación insesgada del vector de medias y la matriz de covarianzas

Ahora consideramos una muestra de un vector de dimensión d, $Y_1,\ldots,Y_n$ i.i.d. con vector de medias $\mu = EY_i$ y matriz de covarianzas $\Sigma = cov(Y_i)$. Los estimadores insesgados de μ y Σ son las versiones multivariantes de la media y varianza muestrales. Si

$$Y_i = \begin{bmatrix} Y_{i1} \\ \vdots \\ Y_{ip} \end{bmatrix}$$

Entonces podemos representar toda la muestra como la siguiente matriz

$$Y = \begin{bmatrix} Y_1' \\ \vdots \\ Y_n' \end{bmatrix} = \begin{bmatrix} Y_{11} & \ldots & Y_{1d} \\ \vdots & \vdots & \vdots \\ Y_{n1} & \ldots & Y_{nd} \end{bmatrix}$$

mientras que los datos observados, la matriz de datos, vendría dada por

$$y = \begin{bmatrix} y_1' \\ \vdots \\ y_n' \end{bmatrix} = \begin{bmatrix} y_{11} & \cdots & y_{1d} \\ \vdots & \vdots & \vdots \\ y_{n1} & \cdots & y_{nd} \end{bmatrix}$$

El vector de medias muestral viene dado por la siguiente expresión en términos de la matriz Y,

$$\bar{Y} = \frac{1}{n}\sum_{i=1}^{n}Y_i = \frac{1}{n}Y'1_n. \tag{20.9}$$

siendo 1_n el vector $n\times 1$ con todos los valores iguales a uno. También denotaremos

$$\bar{Y} = \begin{bmatrix} \bar{Y}_{\cdot 1} \\ \vdots \\ \bar{Y}_{\cdot p} \end{bmatrix}$$

El estimador de la matriz de covarianzas (poblacional) Σ sería la matriz de covarianzas muestral que tiene en la posición (j,k) la covarianza muestral entre las componentes j y k,

$$S_{jk} = \frac{1}{n-1}\sum_{i=1}^{n}(Y_{ij} - \bar{Y}_{\cdot j})(Y_{ik} - \bar{Y}_{\cdot k}),$$

de modo que

$$S = \begin{bmatrix} S_{11} & \dots & S_{1d} \\ \vdots & \vdots & \vdots \\ S_{d1} & \dots & S_{dd} \end{bmatrix} = \frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \bar{Y})(Y_i - \bar{Y})' = \frac{1}{n-1}Q.$$

Es inmediato que $E\bar{Y} = \mu$ porque componente a componente hemos visto que se verifica la igualdad. A partir de los vectores Y_i consideramos $X_i = Y_i - \mu$ de modo que se verifica $\bar{X} = \bar{X} - \mu$. Se sigue que

$$\sum_{i=1}^{n}(Y_i - \bar{Y})(Y_i - \bar{Y})' = \sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})' = \sum_{i=1}^{n} X_i X_i' - n\bar{X}\bar{X}'.$$

Los vectores $X_1, \dots, X_n$ tienen vector de medias nulo y matriz de covarianzas Σ, la misma que los Y_i. En consecuencia, $E\bar{X}\bar{X}' = \Sigma$ y

$$EQ = \sum_{i=1}^{n} cov(Y_i) - n\ cov(\bar{Y}) = n\Sigma - n\ cov(\bar{Y}) = n\Sigma - n\frac{\Sigma}{n} = (n-1)\Sigma.$$

Tenemos pues que S es un estimador insesgado de la matriz Σ.

Finalmente, si denotamos por r_{jk} el coeficiente de correlación entre las variables j y k, es decir,

$$r_{jk} = \frac{\sum_{i=1}^{n}(Y_{ij} - \bar{Y}_{\cdot j})(Y_{ik} - \bar{Y}_{\cdot k})}{\sqrt{\sum_{i=1}^{n}(Y_{ij} - \bar{Y}_{\cdot j})^2 \sum_{i=1}^{n}(Y_{ik} - \bar{Y}_{\cdot k})^2}} = \frac{S_{jk}}{\sqrt{S_{jj}S_{kk}}} \qquad (20.10)$$

Denotaremos por R la **matriz de correlaciones muestrales** $R = [r_{jk}]$.

20.3 Estimador máximo verosímil

El método de estimación que vamos a utilizar en este curso el **método de máxima verosimilitud**. El estimador máximo verosímil de θ, que denotaremos por $\hat{\theta}$, se obtienen máximizando la función de verosimilitud o, equivalentemente, la transformación monótona de dicha función que es la función de logverosimilitud. Utilizaremos para denotar el estimador máximo verosímil la notación inglesa **MLE**.

$$L(\hat{\theta}) = \max_{\theta\in\Theta} L(\theta), \qquad (20.11)$$

o también

$$\hat{\theta} = argmax_{\theta\in\Theta} L(\theta), \qquad (20.12)$$

Ejemplo 20.4 (Bernoulli) *Se puede comprobar sin dificultad que* $\hat{p} = \frac{\sum_{i=1}^{n} x_i}{n}$.

Una propiedad importante de los estimadores máximo verosímiles consiste en que si $\theta^* = f(\theta)$ siendo f una biyección entonces el estimador máximo verosímil de θ^* es verifica que

$$\hat{\theta^*} = f(\hat{\theta}). \tag{20.13}$$

Ejemplo 20.5 (Normal) *En este caso se comprueba que* $\hat{\mu} = \hat{X}_n$ *y que* $\widehat{\sigma^2} = \frac{n-1}{n}S^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X}_n)^2$. *Teniendo en que cuenta la propiedad enunciada en 20.13 tendremos que* $\hat{\sigma} = \sqrt{\frac{n-1}{n}S^2}$.

En muchas situaciones la función $L(\theta)$ es cóncava y el estimador máximo verosímil $\hat{\theta}$ es la solución de las **ecuaciones de verosimilitud** $\frac{\partial L(\theta}{\partial \theta} = 0$. Si $cov(\hat{\theta})$ denota la matriz de covarianzas de $\hat{\theta}$ entonces, para un tamaño muestral grande y bajo ciertas condiciones de regularidad (ver [111], página 364), se verifica que $cov(\hat{\theta})$ es la inversa de la **matriz de información** cuyo elemento (j,k) viene dado por

$$-E\left(\frac{\partial^2 l(\theta)}{\partial\theta_j \partial\theta_k}\right) \tag{20.14}$$

Notemos que el error estándar de $\hat{\theta}_j$ será el elemento que ocupa la posición (j,j) en la inversa de la matriz de información. Cuanto mayor es la curvatura de la logverosimilitud menores serán los errores estándar. La racionalidad que hay detrás de esto es que si la curvatura es mayor entonces la logverosimilitud cae rápidamente cuando el vector θ se aleja de $\hat{\theta}$. En resumen, es de esperar que θ esté más próximo a $\hat{\theta}$.

Ejemplo 20.6 (Binomial) *Supongamos que una muestra en una población finita y consideremos como valor observado el número de éxitos. Entonces la verosimilitud sería*

$$L(p) = \binom{n}{y} p^y (1-p)^{n-y}, \tag{20.15}$$

y la logverosimilitud viene dada como

$$l(p) = \log\binom{n}{y} + y\log p + (n-y)\log(1-p), \tag{20.16}$$

La ecuación de verosimilitud sería

$$\frac{\partial l(p)}{\partial p} = \frac{y}{p} - \frac{n-y}{1-p} = \frac{y-np}{p(1-p)}. \tag{20.17}$$

Igualando a cero tenemos que la solución es $\hat{p} = \frac{y}{n}$ que no es más que la proporción muestral de éxitos en las n pruebas. La varianza asíntótica sería

$$-E\left[\frac{\partial^2 l(p)}{\partial p^2}\right] = E\left[\frac{y}{p^2} + \frac{n-y}{(1-p)^2}\right] = \frac{n}{p(1-p)}. \tag{20.18}$$

En consecuencia asintóticamente $\hat{p}$ tiene varianza $\frac{p(1-p)}{n}$ lo cual era de prever pues si consideramos la variable Y que nos da el número de éxitos entonces sabemos que $EY = np$ y que $var(Y) = np(1-p)$.

	H_0	H_1
Rechazamos H_0	Error tipo I	
No rechazamos H_0		Error tipo II

20.4 Contraste de hipótesis

Genéricamente vamos a considerar situaciones en donde particionamos el espacio paramétrico Θ en dos conjuntos Θ_0 y Θ_1, es decir, $\Theta_0 \cap \Theta_1 = \emptyset$ (son disjuntos) y y $\Theta_0 \cup \Theta_1 = \Theta$ (cubren todo el espacio paramétrico). Consideramos el contraste de hipótesis siguiente.

$$H_0 : \theta \in \Theta_0 \tag{20.19}$$

$$H_1 : \theta \in \Theta_1 \tag{20.20}$$

Basándonos en una muestra aleatoria $X_1, \ldots, X_n$ hemos de tomar una decisión. Las decisiones a tomar son una entre dos posibles: (i) Rechazar la hipótesis nula o bien (ii) no rechazar la hipótesis nula. Notemos que, una vez hemos tomado una decisión, podemos tener dos posibles tipos de error como recoge la siguiente tabla. En las columnas indicamos la realidad mientras que en las filas indicamos la decisión que tomamos.

Supongamos que $\mathbb{R}^n$ es el conjunto de valores que puede tomar el vector aleatorio $(X_1, \ldots, X_n)$. Entonces el contraste de hipótesis se basa en tomar un estadístico o función de la muestra que denotamos $\delta(X_1, \ldots, X_n)$ de modo que si $\delta(X_1, \ldots, X_n) \in C$ entonces rechazamos la hipótesis nula mientras que si $\delta(X_1, \ldots, X_n) \notin C$ entonces no rechazamos la hipótesis nula. Notemos que simplemente estamos particionando el espacio muestral (que suponemos) $\mathbb{R}^n$ en dos partes, C y C^c, de modo que tomamos una decisión basándonos en si el estadístico δ está en C o bien está en el complementario de C. Al conjunto C se le suele llamar la **región crítica**. La función potencia se define como

$$\pi(\theta) = P(\delta \in C | \theta). \tag{20.21}$$

20.4.1 Contraste de la media en la poblaciones normales

Si tenemos una muestra $X_1, \ldots, X_n$ de una población normal con media μ y varianza σ^2 donde ambos parámetros se asumen desconocidos un test habitualmente considerado es si la media toma un valor dado. l test formalmente planteado sería:

$$H_0 : \mu = \mu_0, \tag{20.22}$$

$$H_1 : \mu \neq \mu_0. \tag{20.23}$$

Siendo $S^2 = \frac{\sum_{i=1}^n (X_i - \bar{X})^2}{n-1}$, el estadístico habitualmente utilizado es el siguiente

$$T = \frac{\bar{X} - \mu_0}{S/\sqrt{n}}.$$

Bajo la hipótesis nula este estadístico sigue una distribución t de Student con $n-1$ grados de libertad,

$$T \sim t(n-1).$$

Si suponemos que trabajamos con un nivel de significación α la región crítica en la cual rechazamos la hipótesis nula sería

$$|T| > t_{n-1,1-\frac{\alpha}{2}}.$$

20.4.2 Test del cociente de verosimilitudes

El cociente de verosimilitudes para contrastar estas hipótesis se define como

$$\Lambda = \frac{\max_{\theta\in\Theta_0} L(\theta)}{\max_{\theta\in\Theta} L(\theta)} \tag{20.24}$$

Es razonable pensar que en la medida en que Λ tome valores menores entonces la hipótesis alternativa sea más plausible que la hipótesis nula y por lo tanto rechacemos la hipótesis nula. Realmente se suele trabajar con $-2\log\Lambda$ pues bajo la hipótesis nula tiene una distribución asintótica ji-cuadrado donde el número de grados de libertad es la diferencia de las dimensiones de los espacios paramétricos $\Theta = \Theta_0 \cup \Theta_1$ y Θ_0. Si denotamos $L_0 = \max_{\theta\in\Theta_0} L(\theta)$ y $L_1 = \max_{\theta\in\Theta} L(\theta)$ entonces $\Lambda = \frac{L_0}{L_1}$ y

$$-2\log\lambda = -2\log\frac{L_0}{L_1} = -2(l_0 - l_1) \tag{20.25}$$

siendo l_0 y l_1 los logaritmos de L_0 y L_1 respectivamente que también corresponden con los máximos de la logverosimilitud sobre Θ_0 y sobre Θ.

20.4.3 Test de Wald

Supongamos que el θ es un parámetro y $\hat{\theta}$ denota su estimador máximo verosímil. Supongamos que queremos contrastar las siguientes hipótesis:

$$H_0 : \theta = \theta_0, \tag{20.26}$$
$$H_1 : \theta \neq \theta_0. \tag{20.27}$$

Denotamos por $SE(\hat{\theta})$ el error estándar bajo la hipótesis alternativa de $\hat{\theta}$. Entonces el estadístico

$$z = \frac{\hat{\theta} - \theta_0}{SE(\hat{\theta})} \tag{20.28}$$

tiene, bajo la hipótesis nula, aproximadamente una distribución normal estándar, $z \sim N(0,1)$. Este tipo de estadísticos donde se utiliza el error estándar del estimador bajo la hipótesis alternativa recibe el nombre de *estadístico de Wald*.

Supongamos que θ es un vector de parámetros y queremos contrastar las hipótesis dadas en 20.26. La versión multivariante del estadístico dado en 20.28 viene dada por

$$W = (\hat{\theta} - \theta_0)'[cov(\hat{\theta})]^{-1}(\hat{\theta} - \theta_0), \tag{20.29}$$

donde $cov(\hat{\theta})$ se estima como la matriz de información observada en el MLE $\hat{\theta}$. La distribución asintótica de W bajo la hipótesis nula es una distribución ji-cuadrado donde el número de grados de libertad coincide con el número de parámetros no redundantes en θ.

20.4.4 Intervalos de confianza

Empezamos recordando el concepto de intervalo de confianza con un ejemplo muy conocido como es la estimación de la media en poblaciones normales.

Ejemplo 20.7 (Intervalo de confianza para la media de una normal) *Veámoslo con un ejemplo y luego planteamos la situación más general. Tenemos una muestra aleatoria $X_1, \ldots, X_n$ i.i.d. tales que $X_i \sim N(\mu, \sigma 2)$. Entonces es conocido que*

$$\frac{\bar{X}_n - \mu}{S/\sqrt{n}} \sim t_{n-1}. \tag{20.30}$$

Vemos cómo $\frac{\bar{X}_n - \mu}{S/\sqrt{n}}$ depende tanto de la muestra que conocemos como de un parámetro (la media μ) que desconocemos. Fijamos un valor de α (habitualmente tomaremos $\alpha = 0.05$) y elegimos un valor $t_{n-1,1-\alpha/2}$ tal que

$$P(-t_{n-1,1-\alpha/2} \leq \frac{\bar{X}_n - \mu}{S/\sqrt{n}} \leq t_{n-1,1-\alpha/2}) = 1 - \alpha. \tag{20.31}$$

La ecuación anterior la podemos reescribir como

$$P(\bar{X}_n - t_{n-1,1-\alpha/2}\frac{S}{\sqrt{n}} \leq \mu \leq \bar{X}_n + t_{n-1,1-\alpha/2}\frac{S}{\sqrt{n}}) = 1 - \alpha. \tag{20.32}$$

Tenemos una muestra aleatoria $X_1, \ldots, X_n$ y por lo tanto tenemos un intervalo aleatorio dado por $[\bar{X}_n - t_{n-1,1-\alpha/2}\frac{S}{\sqrt{n}}, \bar{X}_n + t_{n-1,1-\alpha/2}\frac{S}{\sqrt{n}}]$. Este intervalo tiene una probabilidad de $1-\alpha$ de contener a la verdadera media. Tomemos ahora la muestra y consideremos no los valores aleatorios de $\bar{X}_n$ y de S^2 sino los valores observados $\bar{x}_n$ y s. Tenemos ahora un intervalo $[\bar{x}_n - t_{n-1,1-\alpha/2}\frac{s}{\sqrt{n}}, \bar{x}_n + t_{n-1,1-\alpha/2}\frac{s}{\sqrt{n}}]$ fijo. Es posible que μ esté en este intervalo y es posible que no lo esté. Sabemos que antes *de tomar la muestra teníamos una* probabilidad *de $1-\alpha$ de contener a la verdadera media pero* después *de tomar la muestra tenemos una* **confianza** *de $1-\alpha$ de contener a la verdadera media. Al intervalo $[\bar{x}_n - t_{n-1,1-\alpha/2}\frac{s}{\sqrt{n}}, \bar{x}_n + t_{n-1,1-\alpha/2}\frac{s}{\sqrt{n}}]$ se le llama* **intervalo de confianza** *para μ con* **nivel de confianza** $1-\alpha$.

Vamos a ver un planteamiento más general del problema. Supongamos que tenemos un test para contrastar la hipótesis simple $H_0 : \theta = \theta_0$ frente a la alternativa $H_1 : \theta \neq \theta_0$. Supongamos que elegimos un nivel de significación α para contrastar las hipótesis anteriores y consideramos el siguiente conjunto formado por todos los θ_0 tales que no rechazamos la hipótesis nula al nivel α. Este conjunto es un **conjunto de confianza** al nivel $1-\alpha$. Cuando el conjunto de confianza es un intervalo hablamos de **intervalo de confianza**.

Supongamos que consideramos el test del cociente de verosimilitudes. Denotemos por $\chi^2_k(1-\alpha)$ el percentil $1-\alpha$ de una distribución ji-cuadrado con k grados de libertad. Entonces el intervalo de confianza al nivel $1-\alpha$ sería el conjunto

$$\{\theta_0 : -2[l(\theta_0) - l(\hat{\theta})] < \chi^2_k(1-\alpha)\} \tag{20.33}$$

Consideremos ahora un test de Wald. En este caso, el intervalo de confianza de Wald vendría dado por el siguiente conjunto:

$$\{\theta_0 : \frac{|\hat{\theta} - \theta_0|}{SE(\hat{\theta})} < Z_{1-\alpha/2}\} \tag{20.34}$$

donde $SE(\hat{\theta})$ es el error estándar estimado de $\hat{\theta}$ bajo la hipótesis alternativa.

Capítulo 21

Miscelánea

En el texto hemos utilizado una serie de técnicas y conceptos estadísticos. Es preferible, antes que insertar su explicación en el texto incluirlos en un capítulo aparte. Cada sección responde a un concepto distinto. No hay ningún tipo de continuidad entre las secciones. Es un capítulo auxiliar para el resto de los capítulos. Cuando explicamos los conceptos también indicamos cómo hacerlo con R.

21.1 Algoritmo bipesado de Tukey de un solo paso

Su denominación en la literatura estadística sería *One-Step Tukey's Biweight Algorithm*. Veamos cómo funciona. Tenemos unos datos $x_1, \ldots, x_n$ de los cuales pretendemos tener algún tipo de valor medio o de valor central que no esté muy afectado por valores anómalos como lo está, por ejemplo, la media muestral.

1. Calculamos la mediana m_x de los datos $x_1, \ldots, x_n$.
2. Determinamos las distancias $d_i = |x_i - m_x|$.
3. Calculamos la mediana m_d de d_i con $i = 1, \ldots, n$. Esta mediana m_d o mediana de las desviaciones absolutas se suele denotar com MAD (median absolute deviation). Es una medida de dispersión.
4. Consideramos[1]
$$u_i = \frac{x_i - m_x}{cm_d + \epsilon},$$
con $i = 1, \ldots, n$. Los valores por defecto son $c = 5$ y $e = 0.0001$ (con el valor de e evitamos la división por cero).
5. Consideremos una función bicuadrada (bisquare) definida como
$$w(u) = \begin{cases} (1-u^2)^2 & \text{si } |u| \leq 1 \\ 0 & \text{si } |u| > 1 \end{cases}$$
6. Se define el estimador en un solo paso como
$$T_{bi}(x_1, \ldots, x_n) = \frac{\sum_{i=1}^n w(u_i)x_i}{\sum_{i=1}^n w(u_i)}.$$

[1] En la implementación original del procedimiento $c = 1$ y $\epsilon = 0$.

Notemos que si sustituimos el valor m_x por T_b podríamos aplicar iterativamente el procedimiento hasta que se estabilice. No lo hacemos. Simplemente se hace una sola iteración.

Ejemplo 21.1 *Supongamos que tenemos los datos siguientes*

```
x = c(12,3,45,21,34,35,21,456)
```

Y ahora podemos usar la función `tukey.biweight` *del paquete [48, affy].*

```
library(affy)
tukey.biweight(x, c = 5, epsilon = 1e-04)

## [1] 25.1902
```

21.2 Median polish

Tenemos una matriz de datos tal que en la fila i y columna j tenemos y_{ij} con $i = 1, \ldots, I$ y $j = 1, \ldots, J$. El procedimiento del median polish consiste en expresar estos valores de la siguiente forma

$$y_{ij} = m + a_i + b_j + e_{ij}$$

de forma que se verifique que:

- La mediana de $\{a_1, \ldots, a_I\}$ sea cero. Abreviadamente denotamos $Mediana\{a_1, \ldots, a_I\} = 0$.
- $Mediana\{b_1, \ldots, b_J\} = 0$.
- $Mediana\{e_{i1}, \ldots, e_{iJ}\} = 0$ para cada i.
- $Mediana\{e_{1j}, \ldots, e_{Ij}\} = 0$ para cada j.

Pueden haber muchas soluciones para las ecuaciones y restricciones que acabamos de considerar. Consideremos la siguiente matriz.

$$\begin{array}{ccc} y_{11} & \cdots & y_{1J} \\ \vdots & & \vdots \\ y_{I1} & \cdots & y_{IJ} \end{array}$$

El algoritmo median polish (propuesto por Tukey y Mosteller) consiste en aumentar la matriz con una columna y una fila adicionales. Consideremos la matriz aumentada siguiente

$$\begin{array}{ccc|c} e_{11} & \cdots & e_{1J} & a_1 \\ \vdots & & \vdots & \vdots \\ e_{I1} & \cdots & e_{IJ} & a_I \\ \hline b_1 & \cdots & b_J & \mathrm{m} \end{array}$$

El procedimiento actúa iterativamente sobre esta matriz aumentada del siguiente modo:

1. Inicializamos el procedimiento con todos los $a_i = 0$ y $b_j = 0$, $m = 0$ y $e_{ij} = y_{ij}$.

2. Calculamos la mediana de las columnas de la 1 a la I.
3. Restamos dicha mediana a cada elemento de las filas de la 1 a la I y se la sumamos a la columna I+1, esto es, se la sumamos a los valores a_i y m.
4. Calculamos la mediana de cada columna para
5. Restamos dicha mediana a cada elemento de la columna.
6. Sumamos, para cada columna j, la mediana de la columna al valor de la última columna.
7.

Ejemplo 21.2 *Consideremos los datos.*

```
y = matrix(c(18,11,8,21,4,13,7,5,16,7,15,6,7,16,6,19,15,12,18,5),
  ncol=5)
rownames(y) = paste("chip",1:4)
colnames(y) = paste("sonda",1:5)
```

Veamos los datos.

```
y
```

```
##        sonda 1 sonda 2 sonda 3 sonda 4 sonda 5
## chip 1      18       4      16       7      15
## chip 2      11      13       7      16      12
## chip 3       8       7      15       6      18
## chip 4      21       5       6      19       5
```

Y aplicamos una median polish.

```
medpolish(y)
```

```
## 1: 82
## Final: 82
##
## Median Polish Results (Dataset: "y")
##
## Overall: 10.5
##
## Row Effects:
## chip 1 chip 2 chip 3 chip 4
##      5      2     -2     -4
##
## Column Effects:
## sonda 1 sonda 2 sonda 3 sonda 4 sonda 5
##     1.0    -1.5     0.0     0.5    -0.5
##
## Residuals:
##        sonda 1 sonda 2 sonda 3 sonda 4 sonda 5
## chip 1     1.5     -10     0.5      -9       0
## chip 2    -2.5       2    -5.5       3       0
## chip 3    -1.5       0     6.5      -3      10
## chip 4    13.5       0    -0.5      12      -1
```

21.3 Datos faltantes

Tenemos nuestra matriz de datos de expresión X y tenemos datos faltantes en algunas características para algunas muestras. Una posibilidad es no perder datos utilizando alguna técnica que impute un valor a los datos que no tenemos. En § 21.3.1 explicamos el procedimiento utilizado por el método SAM (§ 9.1) y por el método GSA (§ 16.11).

21.3.1 Algoritmo de normalización del k-vecino más próximo

Fijamos el número de vecinos k a utilizar en la imputación. Por defecto se toman $k = 10$. El procedimiento es el siguiente:

1. Para cada gen i con al menos un valor faltante:
   (a) Denotamos por S_i las muestras para los cuales *conocemos* la expresión del gen.
   (b) Determinamos los k-vecinos más próximos del gen i utilizando solamente las muestras S_i para calcular la distancia euclídea. Si los otros genes tienen datos faltantes en las muestras S_i se utilizan las muestras de las cuales se conoce su expresión.
   (c) Para las muestras que no están en S_i asignamos el promedio de los k-vecinos más próximos. Si alguno de estos k-vecinos no tiene alguna expresión entonces hacemos la media de los que disponemos.
2. Si un gen tiene todavía un dato faltante entonces asignamos la expresión media de la muestra considerada.

El algoritmo anterior puede ser lento cuando tenemos muchos genes. Con objeto de acelerar el proceso de asignación de datos faltantes se combina con el siguiente procedimiento de agrupamiento recursivo basado en el k-medias (§ 13.7.1).

1. Si el número de genes p es mayor que p_{max} (por defecto, vale 1500):
   (a) Ejecutamos una clasificación del procedimiento k-medias con k = 2 al conjunto de los genes (clasificamos genes y no muestras). Las distancias entre los genes están basadas en las muestras para las cuales se conocen ambas expresiones.
   (b) Formamos dos matrices de expresión una con cada conjunto de genes. Aplicamos recursivamente el paso anterior.
2. Si $p < p_{max}$ entonces aplicamos el procedimiento de imputación utilizando el promedio de los k-vecinos más próximos en cada uno de los grupos encontrados previamente.

El procedimiento fue propuesto en [135] y está implementado en la función impute::impute.knn.

21.4 Datos multimodales

21.4.1 TCGA

Los datos con los que vamos a trabajar proceden de TCGA.[248]

Podemos utilizar el paquete [78, RTCGA] permite el manejo de datos de esta base de datos de un modo simple.

Lo que sigue está tomado (en gran parte) de la viñeta del paquete [**R-RnaSeqSampleSizeData**].

Los datos se pueden bajar de https://tcga-data.nci.nih.gov/tcga/tcgaDownload.jsp. Por ejemplo, en https://tcga-data.nci.nih.gov/tcgafiles/ftp_auth/distro_ftpusers/anonymous/tumor/brca/cgcc/unc.edu/illuminahiseq_rnaseqv2/rnaseqv2/ se tiene las distintas versiones de las muestras de BRCA. Los conteos están en los ficheros `.rsem.genes.results`.

[248] The Cancer Genome Atlas. Podemos bajar datos con los conteos ya calculados en https://tcga-data.nci.nih.gov/tcga/tcgaHome2.jsp y en https://cghub.ucsc.edu/ podemos conseguir datos originales.

Parte VIII

Investigación reproducible

Capítulo 22

Investigacion reproducible

Tratamos sobre **investigación reproducible**.[249] relacionado con la programación comentada[250] aunque no son sinónimos.

Estamos acostumbrados a que los investigadores[251] en sus publicaciones comenten los resultados obtenidos en sus investigaciones. Han obtenido unos datos, los han analizado y, finalmente, los han interpretado y discutido. *H*emos de creer que han producido los datos correctamente, que han aplicado un tratamiento adecuado (que han hecho lo que dicen que han hecho es un mínimo) y que, finalmente, la interpretación de los resultados de las técnicas utilizadas es correcta. Los demás hemos de creer en ellos. Que los distintos paso se han hecho bien. Al menos, lo mejor que han podido. Y que además el nivel de corrección en todas las etapas es suficiente. Si no tenemos los datos, el código y la interpretación: ¿cómo sabemos que el trabajo es correcto? Es una cuestión de fe. ¿Dónde están los datos? Exactamente, no aproximadamente: ¿qué análisis de los datos han realizado? En otras palabras, deme el código para que yo (y cualquier otra persona) pueda reproducir *exactamente* el tratamiento estadístico que se ha realizado. La interpretación es lo único que tenemos: es la publicación científica que nos ha dado la *n*oticia de la existencia de la investigación.

Por investigación reproducible entendemos procedimientos que permitan *reproducir* la investigación completa. En su totalidad. Y si esto es posible no se debiera de publicar.

Personalmente añadiría que los distintos elementos han de ser totalmente libres. Los datos han de estar a disposición de la comunidad[252] y el software que se utiliza para realizar los análisis han de ser libres también. Quizás esta opción personal es radical pero creo en ella firmemente. [253]

Este manual es un ejemplo de lo que [151] llama un **documento dinámico**. Es un documento en donde se combinan las explicaciones metodológicas junto con el código que las implementa. [254] La herramienta fundamental utilizada es el paquete [152, knitr].[255]. En [151] tenemos una detallada y amena exposición que recomiendo vivamente. Dos son las opciones para escribir el texto, LaTeXy Markdown. Este manual está hecho con LaTeX. Es un documento largo con muchas expresiones matemáticas que hacían aconsejable (imprescindible para un estadístico como yo) utilizar este lenguaje de marcas, el primero de todos y posiblemente el mejor. Sin embargo, los documentos con

[249] Reproducible research

[250] Literate programming

[251] En ciencias experimentales.

[252] Una vez se han publicado los trabajos.

[253] Todo lo que he usado en este manual es libre. No se ha utilizado sofware propietario ni datos que no estén a disposición de todos en algún repositorio público.

[254] Sobre la obtención de los datos hay que consultar las referencias bibliográficas.

[255] Hay miles de paquetes de R pero pocos como este maravilloso paquete.

ejemplos asociados al manual está hechos con otro lenguaje de marcas (ligero) conocido como Markdown. Utilizamos RMarkdown ([5, rmarkdown]).

De un modo genérico son de interés las referencias [58, 57] así como las siguientes direcciones:

1. El task view en el repositorio de R http://cran.r-project.org/web/views/ReproducibleResearch.html.
2. La página de Harrell http://biostat.mc.vanderbilt.edu/twiki/bin/view/Main/StatReport
3. Si se elige la opción de investigación reproducible utilizando LaTeXentonces una (muy) buena referencia [102].

En este documento nos centramos en R y por ello la opción para investigación reproducible es combinar R bien con Markdown bien con LaTeX. Sin embargo, en otros lenguajes hay otras opciones similares y tan buenas como esta. Es recomendable leer `https://blog.ouseful.info/2017/11/15/programming-meh-lets-teach-how-to-write-computational-essays-instead/`.

22.1 Markdown

Es un lenguaje de marcas ligero (minimal sería más correcto). Pretende ser una forma rápida de escribir HTML. Segun su autor, John Gruber, "HTML is a publishing format; Markdown is a writing format."

La dirección indicada contiene una muy buena exposición de este lenguaje de marcas.

De las distintas versiones del lenguaje Markdown la más interesante para nosotros es la asociada a pandoc, **Pandoc's Markdown**.

22.2 RMarkdown

El paquete [5, rmarkdown] incorpora una implementación del lenguaje de marcas Markdown y utilizado conjuntamente con [152, knitr] permite generar unos muy buenos informes. La página `http://rmarkdown.rstudio.com/` es el mejor lugar para aprender a manejarlo.

22.3 RStudio y RMarkdown

Es la mejor opción. RStudio permite elegir esta opción y lleva incorporados en los menús opciones para generar informes en formatos html, pdf e incluso[256] en formado Word.

[256] En el colmo de las locuras.

22.4 emacs y RMarkdown

El editor *emacs* puede ser configurado para trabajar con R y Markdown. Consiste en utilizar el paquete polymode Siguiendo lo indicado en esta página los pasos a seguir (y funciona) son los siguientes:

1. En `/home/gag/emacs.d/`[1] ejecutamos

```
git clone https://github.com/vitoshka/polymode.git
```

[1] Sustituimos por el directorio personal correspondiente.

2. Incluimos en `.emacs` los caminos correspondientes así como los modos necesarios. Hay que añadir las siguientes líneas (sustituyendo el directorio personal por el correspondiente).

```
(add-to-list 'load-path "/home/gag/.emacs.d/polymode/")
(add-to-list 'load-path "/home/gag/.emacs.d/polymode/modes/")
(require 'poly-R)
(require 'poly-markdown)
```

Capítulo 23

Lo que no se debe hacer pero se hace

En un texto de Estadística aplicada a datos ómicos parece que lo natural es explicar solamente cómo diseñar un experimento y cómo analizar de un modo correcto los datos obtenidos. Pero la experiencia le dice a uno que raramente preguntan los experimentadores previamente a realizar su estudio. Luego no suelen encontrar a algún alma que les intente analizar los datos obtenidos sin pensar un poco en el diseño previo. Esto es así.

En este tema hablamos de barbaridades. Pequeñas y grandes barbaridades. Posiblemente las más peligrosas son las pequeñas por lo frecuentes. Ejemplos que me he ido encontrando de lo que se hace y (en mi opinión) no se debe hacer. Responde a una experiencia personal. Vamos a explicar en negativo (lo incorrecto) y no en positivo (lo correcto) como se pretende en el resto del texto. No es una mala opción. Si todo el texto se hiciera así sería muy largo ya que el número de incorrecciones que se pueden cometer es infinito.

Esperemos que, al menos, sea un infinito numerable. Pero no lo sé. Podría ser de un cardinal estrictamente mayor. Yo diría que con probabilidad uno lo es.

23.1 ¿Variables u observaciones? ¿Qué estoy analizando?

Solo Dios lo sabe. En datos ómicos se tienen muchas variables y pocas observaciones. Existe una tradición (extraña para mi) de disponer los datos (de expresión y no solo de expresión) en una forma no habitual en Estadística. En concreto en la matriz de expresión tenemos en **filas** las variables (correspondientes a sondas, genes, isoformas, proteínas, etc.) y en columnas tenemos las observaciones. Pero no hay que olvidar lo que son observaciones y lo que son las variables. Esto parece obvio pero no lo es. Además tenemos la desgracia de que estamos observando una misma característica pero en distintos entes biológicos. Esto hace que el avispado (Dios le perdone) compare libremente dentro de una misma observación (columna de la matriz de expresión) diferentes conjuntos de valores correspondientes a distintos conjuntos de filas (por ejemplo, distintos grupos de genes). Utilizando lo que Dios le da a entender o que les sale significativo. Más bien lo segundo porque lo único que suele tener interés es que sea **significativo** (que hay que publicar un artículo al mes). [257]

[257] Personalmente se lo he intentado explicar a algún catedrático de Bioquímica pero como el que oye llover. Además el argumento de la respuesta es brillante -Pero si me lo ha publicado en una

23.2 Mejor no usarlo

Un breve listado de cosas que hay que evitar.

Diagramas de sectores Los periódicos los aman. Los experimentadores también. Y no lo entiendo. Lo más habitual entre los estadísticos es odiarlos. En mi caso con toda mi alma. No usarlos. Es mucho más claro un diagrama de barras como opción alternativa si queremos un dibujo de frecuencias. La comparación entre las frecuencias de las distintas categorías es mucho más simple. Evitarlos.

Histogramas Los histogramas como una estimación de la función de densidad de la variable (aleatoria) que estamos estudiando tienen una gran tradición en Estadística. Su principal problema es que la elección del número de clases es crítica. El dibujo cambia mucho cuando modificamos el número de clases. Depende demasiado de una elección previa del usuario. Los programas llevan procedimientos que fijan este número. Es mejor utilizar un estimador no paramétrico de la densidad o estimadores basados en funciones kernel. En el texto lo usamos. Con grandes cantidades de datos dan una muy buena (y robusta) estimación de la función de densidad de la variable aleatoria.

23.3 Combinando información de distintos experimentos

Las (buenas) revistas científicas han introducido el buen hábito de obligar a que se depositen los datos en repositorios público. Este texto utiliza datos de estos repositorios. Esto es muy bueno. Además debiera de obligarse a poner el código que se ha desarrollado en el lenguaje o lenguajes que sean para poder reproducir el estudio. Este último punto todavía no es habitual pero creo que lo será en no mucho tiempo. Al menos en buenas revistas. Tener muchos bancos de datos disponibles es bueno. Pero puede ser malo, de hecho, puede ser malísimo si no se usan con sensatez. Ha aparecido otra bibliografía que utiliza estos datos *prestados* para realizar análisis conjuntos o **meta-análisis**. El meta-análisis es una parte de la Estadística que se explica poco. No suele estar en los programas de las asignaturas básicas. Y podría estar. Y debiera de estar porque se están cometiendo auténticos crimenes (contra la humanidad).

En datos ómicos hay una cierta confusión entre qué son las variables y qué son las observaciones. Y a esta confusión no ayuda el costumbre de representar las matrices de expresión traspuestas. La matriz de datos tiene habitualmente las filas correspondiendo con las observaciones y las columnas correspondiendo a las variables que observamos en dichas observaciones. Sin embargo, con datos ómicos es muy habitual (y en este texto seguimos este hábito) representar la matriz de expresión de modo que las columnas son observaciones o muestras y las filas son las variables (genes, sondas, grupos de sondas, etc). Las columnas o muestras u observaciones son independientes mientras que las filas no lo son. Es conveniente seguir esta costumbre porque los paquetes de Bioconductor lo hacen y son nuestra fuente principal de software estadístico.

Cuando combinamos información de distintos experimentos es claro que las variables pueden parecer ser las mismas pero no serlo. Una variable puede estar cuantificando información sobre un mismo gen pero tiene porqué tener una misma distribución de probabilidad. No tenemos muestras aleatorias en donde se asume una distribución común. Por ejemplo, nos colocamos en la mejor de las situaciones. Cada experimento utiliza el mismo chip. ¿Podemos simplemente combinar las columnas y metadatos? Los distintos experimentos se han realizado en distintos laboratorios. ¿Sus protocolos experimentales son los mismos? ¿El material biológico se ha seleccionado de la misma forma? ¿Hay influencia del laboratorio? Son preguntas que si tienen respuesta negativa significa que no observamos lo mismo aunque sea el mismo chip y el mismo diseño.

Por supuesto si el chip no es el mismo no tiene sentido combinar la información considerando que tenemos una muestra mayor y quedándonos con aquellos genes que están en todos los chips. Aunque hablamos de unos mismos genes no tenemos la misma variable aleatoria. Si la covariable que utilizamos para evaluar los perfiles de expresión es la **misma** entonces podemos pensar en realizar un meta-análisis utilizando los p-valores pero no combinando efectos. No tenemos la misma variable.

¿Y con RNA-seq? La cosa no es tan simple de evaluar. Es claro que el tema de los protocolos han de ser los mismos. Y luego hay que utilizar los mismos alineadores y con los mismos parámetros. Hay que considerar los posibles errores sistemáticos producidos por los procedimientos de alineamiento. En este sentido si se utilizan datos procesados disponibles en la red no parece razonable combinar esta información. Se debe de utilizar el mismo material base, con lecturas de la misma longitud, conseguidas con el mismo producto.

Una práctica (que espero no se transforme en habitual) consiste en construir bancos de datos reunión datos de distintos experimentos.

Esta confusión hace que a veces se toman distintos experimentos. De cada uno de estos experimentos se observa una variable distinta asociada a un mismo gen. Y se construye una matriz de datos donde en filas tenemos genes y en columnas tenemos variables medidas en distintos experimentos. Las variables que observamos en distintas experimentaciones aunque se refieran a un mismo gen son observaciones independientes que además no están observadas en las mismas condiciones experimentales. Por lo tanto no podemos estudiar la dependencia que pueda existir entre ellas. Insisto, son valores independientes de distribuciones distintas y la posible dependencia que observemos será puro ruido. Y se hace.

Apéndice A

De cómo se resuelven algunos de los problemas propuestos

Solución (Ej. 1) —

1.
```
mean(x)
```
```
## [1] 5
```
```
var(x)
```
```
## [1] 4.484848
```
```
sd(x)
```
```
## [1] 2.117746
```

2.
```
sum(x == 3)
```
```
## [1] 16
```

3.
```
y = sort(x)
cumsum(y)
```
```
##   [1]   0   1   3   5   7   9  11  13  15  17  20
##  [12]  23  26  29  32  35  38  41  44  47  50  53
##  [23]  56  59  62  65  69  73  77  81  85  89  93
##  [34]  97 101 105 109 113 117 121 125 129 133 137
##  [45] 141 146 151 156 161 166 171 176 181 186 191
##  [56] 196 201 206 211 216 221 226 232 238 244 250
##  [67] 256 262 268 274 280 286 292 298 304 310 316
##  [78] 323 330 337 344 351 358 365 372 379 387 395
##  [89] 403 411 419 427 435 443 452 461 470 479 489
## [100] 500
```

Solución (Ej. 2) —

1.
```
sum(x >= 39.4)
```
```
## [1] 23328
```

O bien podemos ver los `TRUE` de

```
table(x >= 39.4)
```

```
##
## FALSE  TRUE
##   161 23328
```

2.
```
sum(x <46)
```

```
## [1] 15778
```

3.
```
sum(x >= 39.4 & x < 46)
```

```
## [1] 15617
```

4. Significa que son mayores o iguales a 40 y estrictamente menores que 41. Por tanto podemos

```
sum(x >= 40 & x < 41)
```

```
## [1] 638
```

Otra posibilidad es usar la función base::floor.

```
sum(floor(x) == 40)
```

```
## [1] 638
```

Solución (Ej. 3) —

1.
```
ncol(x)
```

```
## [1] 122
```

2.
```
which.max(apply(x,2,mean))
```

```
## [1] 58
```

3.
```
which.min(apply(x,1,mean))
```

```
## [1] 143
```

4.
```
which.min(apply(x,1,IQR))
```

```
## [1] 282
```

5.
```
sum(x[,1] >= x[,2])
```

```
## [1] 1183
```

6.
```
rowmean = apply(x,1,mean)
sum(x[,45] >= rowmean)
```

```
## [1] 1220
```

Solución (Ej. 4) — Empezamos leyendo los datos y transformando en `factor` la variable `golub.cl`.

```
data(golub,package="multtest")
golub.fac = factor(golub.cl,levels=0:1,labels=c("ALL","AML"))
```

1. La expresión media en cada condición la tenemos con

```
m1 = apply(golub,1,mean)
```

2.
```
sd1 = apply(golub,1,sd)
```

3.
```
df = data.frame(x0= m1,y0 =sd1)
pdf(paste0(dirTamiFigures,"tt171113a.pdf"))
ggplot(df,aes(x=x0,y=y0)) + geom_point()
dev.off()
```

Solución (Ej. 7) — 1.
```
ncol(x)
```
```
## [1] 5
```

2.
```
apply(x,2,max)
```

3.
```
minrow = apply(x,1,min)
which.min(minrow)
```
```
## [1] 78
```

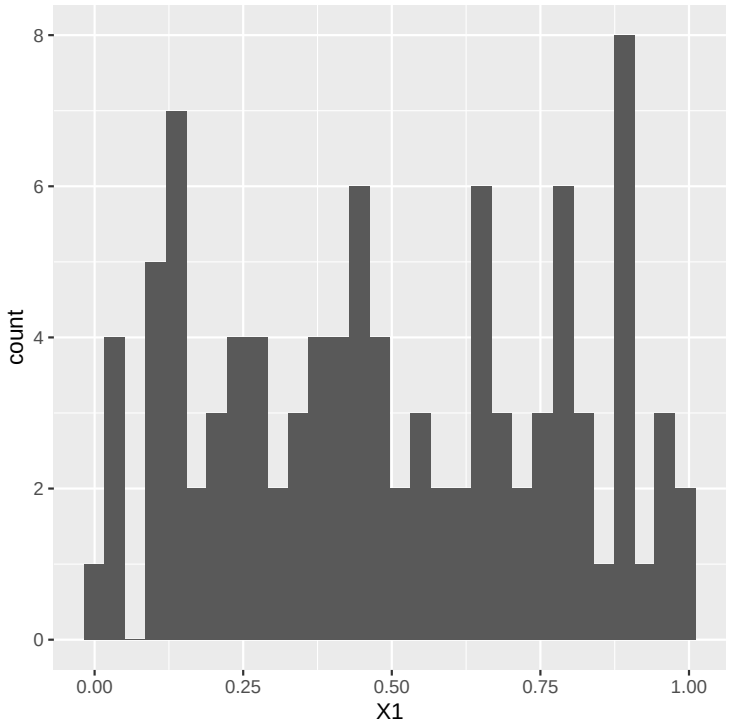
4.
```
which.max(minrow)
```
```
## [1] 50
```

5.
```
quantile(x[,1],probs=0.34)
```
```
##       34%
## 0.3489822
```

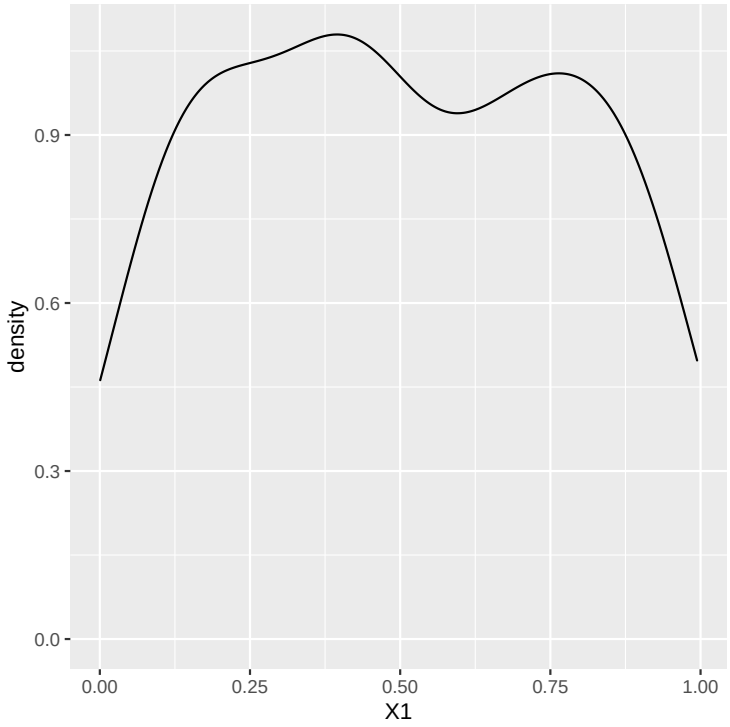
6.
```
meanrow = apply(x,1,mean)
quantile(x[,1],probs=0.34)
```
```
##       34%
## 0.3489822
```

7.
```
min(x)
```
```
## [1] 0.0004653491
```
```
max(x)
```
```
## [1] 0.9994045
```
```
mean(x)
```
```
## [1] 0.4952837
```

8.
```
pacman::p_load(ggplot2)
df= data.frame(x)
ggplot(df,aes(x=X1)) + geom_histogram()
```



9.
```
pacman::p_load(ggplot2)
df= data.frame(x)
ggplot(df,aes(x=X1)) + geom_density()
```



10.
```
sum(x>.12)
```
```
## [1] 448
```

11.
```
x1 = apply(x< .12,1,sum)
```

12.
```
IQR(x1)
```
```
## [1] 1
```

13.
```
sum(x[,1] > x[,2])
```
```
## [1] 51
```

Solución (Ej. 8) —
1.
```
data(golub,package="multtest")
df = t(golub)
df = data.frame(df)
```

2.
```
names(df) = paste0("probe",1:ncol(df))
```

3.
```
type = factor(golub.cl,levels=0:1,labels=c("ALL","AML"))
df= data.frame(df,type)
```

Solución (Ej. 9) —
```
a100218a = function(x)
  list(media = mean(x),varianza = var(x))
```

Solución (Ej. 10) —
```
a100218a = function(x){
    q0 = quantile(x,probs = c(alpha,1-alpha))
    sel0 = which(x >= q0[1] &  x <= q0[2])
    list(media_ajustada = mean(x[sel0]),varianza_ajustada = var(x[sel0]))
    }
```

Solución (Ej. 11) — Los dos primeros paquetes [67, ggfortify] y [45, faraway] están en CRAN y por lo tanto los instalamos con utils::install.packages. Nos pedirá que elijamos repositorio. Elegimos el que nos guste.

```
pkgs = c("ggfortify","faraway")
install.packages(pkgs)
```

Los paquetes [108, multtest] y [33, edgeR] son de Bioconductor. Se instalan con

```
source("https://bioconductor.org/biocLite.R")
pkgs = c("multtest","edgeR")
biocLite(pkgs)
```

Solución (Ej. 12) — Cargamos la base de datos de tipo OrgDb correspondiente a homo sapiens.

```
pacman::p_load(org.Hs.eg.db)
```

Buscamos cómo acceder al SYMBOL o nombre abreviado del gen.

```
keytypes(org.Hs.eg.db)
##  [1] "ACCNUM"       "ALIAS"        "ENSEMBL"
##  [4] "ENSEMBLPROT"  "ENSEMBLTRANS" "ENTREZID"
##  [7] "ENZYME"       "EVIDENCE"     "EVIDENCEALL"
## [10] "GENENAME"     "GO"           "GOALL"
## [13] "IPI"          "MAP"          "OMIM"
## [16] "ONTOLOGY"     "ONTOLOGYALL"  "PATH"
## [19] "PFAM"         "PMID"         "PROSITE"
## [22] "REFSEQ"       "SYMBOL"       "UCSCKG"
## [25] "UNIGENE"      "UNIPROT"
```

Por tanto los nombres abreviados de los genes los tendremos con

```
keys(org.Hs.eg.db, keytype="SYMBOL")
```

Una opción muy simple puede ser

```
(posBRCA1 = which(keys(org.Hs.eg.db, keytype="SYMBOL") == "BRCA1"))
## [1] 553
(posBRCA2 = which(keys(org.Hs.eg.db, keytype="SYMBOL") == "BRCA2"))
## [1] 555
posBRCA = c(posBRCA1,posBRCA2)
```

Supongamos que no tenemos claro su nombre exacto y sabemos que es BRCA seguido de algo más. Podemos mirar aquellos genes que empiezan por BRCA.

```
(beginBRCA = grep("^BRCA",keys(org.Hs.eg.db, keytype="SYMBOL")))
## [1]   553   555  6386 14079 23827
```

¿Cuáles son sus nombres abreviados?

```
keys(org.Hs.eg.db, keytype="SYMBOL")[beginBRCA]
## [1] "BRCA1"   "BRCA2"   "BRCATA"  "BRCA3"
## [5] "BRCA1P1"
```

Construimos un data.frame con la información pedida.

```
symbol = keys(org.Hs.eg.db, keytype="SYMBOL")[posBRCA]
entrezid =  keys(org.Hs.eg.db, keytype="ENTREZID")[posBRCA]
ensembl =  keys(org.Hs.eg.db, keytype="ENSEMBL")[posBRCA]
go =  keys(org.Hs.eg.db, keytype="GO")[posBRCA]
data.frame(symbol, entrezid,ensembl,go)
##   symbol entrezid         ensembl         go
## 1  BRCA1      672 ENSG00000206372 GO:0042136
## 2  BRCA2      675 ENSG00000226560 GO:0042982
```

Solución (Ej. 13) — `pacman::p_load(org.Mm.eg.db)`

```
(posBRCA1 = which(keys(org.Mm.eg.db, keytype="SYMBOL") == "BRCA1"))
## integer(0)
```

Solución (Ej. 21) — 1.Seleccionamos:

```
data(gse20986,package="tamidata")
```

¿Qué columnas ocupan?

```
pacman::p_load("Biobase")
pData(gse20986)[,"tissue"] == "iris" | pData(gse20986)[,"tissue"] == "huvec"
```

De otro modo:

```
(sel = which(is.element(pData(gse20986)[,"tissue"],c("iris","huvec"))))
## [1]  1  4  6 10 11 12
```

Seleccionamos las muestras.

```
eset = gse20986[,sel]
```

Con el siguiente código eliminamos las categorías que no utilizamos.

```
pData(eset)[,"tissue"] = factor(pData(eset)[,"tissue"])
```

2.Con genefilter::rowttests.

```
tt = genefilter::rowttests(eset,pData(eset)[,"tissue"])
```

3.

```
tt1 = multtest::mt.rawp2adjp(tt[,"p.value"],"BH")
```

Los p-valores ajustados serían

```
tt1$adjp[,2]
```

4.Tenemos como significativos

```
sum(tt1$adjp[,2] < 0.05,na.rm = TRUE)
```

```
## [1] 1828
```

que ocupan las siguientes posiciones en la matriz original.

```
sigIH = tt1$index[1:1828]
```

5.
```
sel = which(is.element(pData(gse20986)[,"tissue"],c("retina","huvec")))
eset = gse20986[,sel]
pData(eset)[,"tissue"] = factor(pData(eset)[,"tissue"])
tt = genefilter::rowttests(eset,pData(eset)[,"tissue"])
tt1 = multtest::mt.rawp2adjp(tt[,"p.value"],"BH")
sigRH = tt1$index[1:sum(tt1$adjp[,2] < 0.05,na.rm = TRUE)]
```

6.
```
sel = which(is.element(pData(gse20986)[,"tissue"],c("coroides","huvec")))
eset = gse20986[,sel]
pData(eset)[,"tissue"] = factor(pData(eset)[,"tissue"])
tt = genefilter::rowttests(eset,pData(eset)[,"tissue"])
tt1 = multtest::mt.rawp2adjp(tt[,"p.value"],"BH")
sigCH = tt1$index[1:sum(tt1$adjp[,2] < 0.05,na.rm = TRUE)]
```

7.
```
sigIR = intersect(sigIH,sigRH)
sigIRC = intersect(sigIR,sigCH)
```

Y el resultado (no mostrado) es

```
sigIRC
```

Que corresponde con los genes.

```
featureNames(gse20986[sigIRC,])
```

Solución (Ej. 22) — Leemos los datos.

```
data(gse1397,package="tamidata")
eset = gse1397
```

1.
```
tt = genefilter::rowttests(eset,pData(eset)[,"type"])
```

Solución (Ej. 25) —

1.
```
pacman::p_load("SummarizedExperiment","GenomicRanges","edgeR","
data(PRJNA297664,package="tamidata")
se = PRJNA297664
rm(PRJNA297664)
pvalor13 = binomTest(assay(se)[,1], assay(se)[,3])
```

2.
```
tt = multtest::mt.rawp2adjp(pvalor13,"BH")
pvalor13.BH = tt$adjp["BH"]
```

3.
```
numsig = sum(pvalor13.BH < 0.001)
sig13 = rownames(se)[tt$index[1:numsig]]
```

4.
```
pvalor25 = binomTest(assay(se)[,2], assay(se)[,5])
tt = multtest::mt.rawp2adjp(pvalor25,"BH")
pvalor25.BH = tt$adjp[,"BH"]
numsig = sum(pvalor25.BH < 0.001)
sig25 = rownames(se)[tt$index[1:numsig]]
```

5.
```
intersect(sig13,sig25)
```

Apéndice B

Todo lo que siempre quiso saber sobre R pero nunca se atrevió a preguntar

El título lo dice. ¿Para qué insistir? R es excesivo en todos los sentidos. Personalmente estoy un poco harto de R. Pero es lo que hay. Es muy útil y tenemos implementado casi todo lo que a mi se me ocurre que tiene interés. Pues eso. Más R.

B.1 Pasar argumentos a un script de R por línea de comandos

[258]

El script de R debe comenzar de la siguiente manera:

[258] Luis Orduña me mostró cómo hacerlo.

```
args = commandArgs(trailingOnly=TRUE)
if (length(args)==0) {
  stop("At least one argument must be supplied (path to the folder
that contains raw_data).", call.=FALSE)
}
```

Para ejecutar el script pasándole argumentos simplemente hay que hacer

```
Rscript --vanilla script.R argument
```

Se puede encontrar información adicional en https://www.r-bloggers.com/passing-arguments-to-an-r-script-from-command-lines/.

B.2 tamitasks

¿Cómo hacer un paquete en la medida en que vamos desarrollando un proyecto? ¿Cómo hacer un paquete sin complicarse mucho la vida? Trabajar con paquetes en R es muy importante. Permite organizar y

comprobar que estamos realizando un trabajo correctamente. Desde el principio hay que aprender a construir nuestro propio paquete. De esto hablamos aquí. Lo que comentamos en esta sección está ejemplificado en el paquete `tamitasks` que podemos encontrar en http://www.uv.es/ayala/docencia/tami/tamitasks

Una muy buena referencia sobre cómo construir un paquete es [140]. Lo que sigue es una breve forma de construir un paquete muy sencillo en Linux.[259]

[259] He de reconocer que no tengo el más mínimo interés en saber cómo se hace en Windows. No al menos en esta vida.

1. Hemos de crear un directorio con el nombre del paquete.

```
mkdir tamitasks
```

2. Luego creamos directorios dentro de `tamitasks`.

```
cd tamitasks
mkdir R
mkdir data
mkdir man
mkdir doc
mkdir vignettes
```

3. Creamos en `tamitasks` el fichero `DESCRIPTION`. Un posible ejemplo puede ser

```
Package: tamitasks
Type: Package
Title: tamitasks
Version: 0.2
Date: "`r Sys.Date()`"
Authors@R: person("Guillermo","Ayala",email="Guillermo.Ayala@uv.es",
                  role = c("aut","cre"))
Description: A (very) basic package
Imports:
    AnnotationDbi,
    annotate,
    affy,
    Biobase,
    DBI,
    edgeR,
    genefilter,
    ggfortify,
    ggplot2,
    gridExtra,
    GSA,
    GSEABase,
    GEOquery,
    hgu133plus2.db,
    limma,
    MASS,
    matrixStats,
    methods,
    multtest,
    pacman,
    pbapply,
```

```
    qvalue,
    ReportingTools,
    reshape2,
    samr,
    stats,
    SummarizedExperiment,
    tamidata
Suggests: knitr,
        rmarkdown
URL: http://www.uv.es/ayala/docencia/tami
VignetteBuilder: knitr
License: GPL-2
LazyLoad: yes
RoxygenNote: 7.1.0
```

4. Copiamos en `data` los ficheros binarios de datos. En mi caso son los ficheros `geod71810.rda` y `PRJNA297664.rda`.

5. Copiamos en `vignettes` las viñetas. En mi caso son las viñetas en donde se muestra cómo se han construido los datos del punto anterior. Son las viñetas `geod71810.Rmd` y `PRJNA297664.Rmd`.

6. Copiamos en el subdirectorio `tamitasks/R` los ficheros con el código en R. En mi caso son los ficheros `tasks-data.R` y `tasks-functions.R`. En `tasks-data.R` se muestra cómo definir datos para poder cargarlos.

```
#' @title
PRJNA297664
@description
Saccharomyces cerevisiae
Expression profiling by high throughput
    ↪ sequencing
Identification of the difference in transcriptome
    ↪  between wild-type and
SEC66 deletion mutant cells
@source
\url{http://www.ncbi.nlm.nih.gov/bioproject
    ↪ /297664}
\url{http://www.ncbi.nlm.nih.gov/geo/query/acc.
    ↪ cgi?acc=GSE73681}
The procedure to produce the counts per gene from
    ↪  the original sra files
can be found at
\url{http://www.uv.es/ayala/docencia/tami/
    ↪ CaseStudies/PRJNA297664.html}
@examples
data(PRJNA297664,package="tamitasks")
@docType data
@keywords datasets
@format SummarizedExperiment
@name PRJNA297664
NULL


geod71810
```

```
@description
ExpressionSet \code{geod71810}
@examples
data(geod71810,package="tamitasks")
@docType data
@keywords datasets
@format ExpressionSet
@name geod71810
NULL
```

En `tasks-functions.R` tenemos un par de ejemplos de funciones.

```
@title Trapezoid rule for numerical integration
@description
An implementation of the trapezoid rule for
    ↪ numerical integration
@param x Abscisas
@param y Function values at \code{x}
@export
TrapezoidRule = function(x,y){
  idx = 2:length(x)
  return (as.double( (x[idx] - x[idx-1]) %*% (y[
      ↪ idx] + y[idx-1])) / 2)
}

URL's from entrez identifiers
 @description
 It makes the url's from entrez identifiers
@param id ENTREZ identifiers
@return
The corresponding URL's in the database
 @export
 @family URL generation
 entrezid2url = function(id)
 ifelse(id == "NA",NA,
 paste("<a href='http://www.ncbi.nlm.nih.gov/gene
 ↪ /?term=",
 id,"'>",id,"</a>",sep=""))
```

Como podemos ver en los ejemplos anteriores documentamos el paquete utilizando el paquete [144].

7. Ya tenemos hecho el paquete y hemos de construirlo. Empezamos con algo de código de fácil comprensió.

```
pkg = "tamitasks"
version = "0.2"
dirbasepkg = "/home/gag/ownCloud/alltami/"
dirpkg = paste0(dirbasepkg,pkg)
dirvignettes = paste0(dirbasepkg,pkg,"-notes/
    ↪ vignettes/")
pkgtar = paste0(pkg,"_",version,".tar.gz")
```

8. Cargamos paquetes necesarios.

```
pacman::p_load(devtools,pkgdown)
```

9. Fijamos directorio para construir el paquete.

```
setwd(dirbasepkg)
```

10. Generamos la documentación con [144].

```
devtools::document(pkg)
```

11. Vemos si tenemos algún error.

```
devtools::check(pkg)
```

12. Y construimos el paquete. Tenemos dos opciones:

    (a) Utilizando `devtools::build()`.

    ```
    devtools::build(pkg)
    ```

    (b) Directamente utilizando la línea de comandos.

    ```
    R CMD build --resave-data tamitasks
    ```

13. Instalamos el paquete.

```
install.packages(paste0(pkg,"_",version,".tar.gz"
    ↪ ),repos=NULL,source=TRUE)
```

14. Generamos la página web asociada. Queda bonito.

```
library(tamitasks)
setwd(dirpkg)
pkgdown::build_site()
```

15. Si después de todo lo previo funciona has tenido suerte. Si no empieza desde el primer punto y mira en qué te has equivocado. Creo que esto es la (Bio)Informática.

B.3 Actualizar paquetes

Es frecuente que tengamos que actualizar todos los paquetes que tenemos instalados. Con el siguiente código lo podemos hacer.

```
install.packages(
    lib  = lib <- .libPaths()[1],
    pkgs = as.data.frame(installed.packages(lib),
                         stringsAsFactors=FALSE)$Package,type = 'source'
)
```

Bibliografía

[1] H. Abdi y L. J. Williams. «Principal component analysis». En: *Wiley Interdisciplinary Reviews: Computational Statistics* 2.4 (2010), págs. 433-459. URL: /home/gag/BIBLIOGRAFIA/REPRINTS/AbdiWilliams10.pdf.

[2] Sudipta Acharya, Sriparna Saha y N. Nikhil. «Unsupervised gene selection using biological knowledge : application in sample clustering». En: *BMC Bioinformatics* 18.1 (2017), pág. 513. ISSN: 1471-2105. DOI: 10.1186/s12859-017-1933-0. URL: https://doi.org/10.1186/s12859-017-1933-0.

[3] A. Agresti. *Categorical Data Analysis*. Second. Wiley, 2002. URL: /home/gag/BIBLIOGRAFIA/MISLIBROS/Agresti_Alan_Categorical_Data_Analysis_2nd_ed_2002.pdf.

[4] Adrian Alexa y Jorg Rahnenfuhrer. *topGO: Enrichment Analysis for Gene Ontology*. R package version 2.34.0. 2018.

[5] JJ Allaire y col. *rmarkdown: Dynamic Documents for R*. R package version 1.11. 2018. URL: https://CRAN.R-project.org/package=rmarkdown.

[6] S. Anders y col. «Count-based differential expression analysis of RNA sequencing data using R and Bioconductor». En: *Nat. Protocols* 8.9 (2013), págs. 1765-1786. DOI: 10.1038/nprot.2013.099.

[7] Simon Anders y Wolfgang Huber. «Differential expression analysis for sequence count data». En: *Genome Biology* 11.10 (2010), R106. ISSN: 1465-6906. DOI: 10.1186/gb-2010-11-10-r106. URL: http://genomebiology.com/2010/11/10/R106.

[8] John D. Storey with contributions from Andrew J. Bass, Alan Dabney y David Robinson. *qvalue: Q-value estimation for false discovery rate control*. R package version 2.14.0. 2018. URL: http://github.com/jdstorey/qvalue.

[9] Guillermo Ayala. *tami: Statistical Bioinformatics*. R package version 0.9. 2019. URL: http://www.uv.es/ayala/docencia/tami.

[10] Guillermo Ayala. *tamidata: Data sets for Statistical Bioinformatics*. R package version 0.5. 2019. URL: http://www.uv.es/ayala/docencia/tami.

[11] M. Baker y D. Penny. «Is there a reproducibility crisis?» En: *Nature* 533.7604 (2016). cited By 44, págs. 452-454. DOI: 10.1038/533452A. URL: http://www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-1.19970.

[12] Andrew J. Bass y col. *biobroom: Turn Bioconductor objects into tidy data frames*. R package version 1.14.0. 2018. URL: https://github.com/StoreyLab/biobroom.

[13] Yoav Benjamini y Yosef Hochberg. «Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing». English. En: *Journal of the Royal Statistical Society. Series B (Methodological)* 57.1 (1995), págs. 289-300. ISSN: 00359246. URL: http://www.jstor.org/stable/2346101.

[14] Yoav Benjamini y Daniel Yekutieli. «The control of the false discovery rate in multiple testing under dependency». En: *The Annals of Statistics* 29.4 (2001), págs. 1165-1188.

[15] J. M. Bland y D. G. Altman. «Statistical methods for assessing agreement between two methods of clinical measurement». En: *Lancet* (1986), págs. 307-310. URL: http://www-users.york.ac.uk/~mb55/meas/ba.htm.

[16] B.M. Bolstad y col. «A comparison of normalization methods for high density oligonucleotide array data based on variance and bias». En: *Bioinformatics* 19.2 (2003), págs. 185-193. DOI: 10.1093/bioinformatics/19.2.185. eprint: http://bioinformatics.oxfordjournals.org/content/19/2/185.full.pdf+html. URL: http://bioinformatics.oxfordjournals.org/content/19/2/185.abstract.

[17] Ben Bolstad. *affyPLM: Methods for fitting probe-level models*. R package version 1.58.0. 2018. URL: https://github.com/bmbolstad/affyPLM.

[18] Ben Bolstad. «Methods in Microarray Normalization». En: Drug Discovery Series 10. CRC Press, 2008. Cap. 3, págs. 41-60.

[19] Anne-Laure Boulesteix y Martin Slawski. «Stability and aggregation of ranked gene lists». En: *Briefings in Bioinformatics* 10.5 (2009), págs. 556-568. DOI: 10.1093/bib/bbp034. eprint: http://bib.oxfordjournals.org/content/10/5/556.full.pdf+html. URL: http://bib.oxfordjournals.org/content/10/5/556.abstract.

[20] Richard Bourgon, Robert Gentleman y Wolfgang Huber. «Independent filtering increases detection power for high-throughput experiments». En: *Proceedings of the National Academy of Sciences* 107.21 (2010), págs. 9546-9551. DOI: 10.1073/pnas.0914005107. eprint: http://www.pnas.org/content/107/21/9546.full.pdf+html. URL: http://www.pnas.org/content/107/21/9546.abstract.

[21] Rosemary Braun, Leslie Cope y Giovanni Parmigiani. «Identifying differential correlation in gene/pathway combinations». En: *BMC Bioinformatics* 9.1 (2008), pág. 488. ISSN: 1471-2105. DOI: 10.1186/1471-2105-9-488. URL: http://www.biomedcentral.com/1471-2105/9/488.

[22] Andrew C Browning y col. «Comparative gene expression profiling of human umbilical vein endothelial cells and ocular vascular endothelial cells». En: *British Journal of Ophthalmology* 96.1 (2012), págs. 128-132. DOI: 10.1136/bjophthalmol-2011-300572. eprint: http://bjo.bmj.com/content/96/1/128.full.pdf+html. URL: http://bjo.bmj.com/content/96/1/128.abstract.

[23] M. Carlson y col. *GenomicFeatures: Tools for making and manipulating transcript centric annotations*. R package version 1.34.1. 2018.

[24] Marc Carlson. *ath1121501.db: Affymetrix Arabidopsis ATH1 Genome Array annotation data (chip ath1121501)*. R package version 3.2.3. 2016.

[25] Marc Carlson. *GO.db: A set of annotation maps describing the entire Gene Ontology*. R package version 3.7.0. 2018.

[26] Marc Carlson. *hgu133a.db: Affymetrix Human Genome U133 Set annotation data (chip hgu133a)*. R package version 3.2.3. 2016.

[27] Marc Carlson. *hgu95av2.db: Affymetrix Human Genome U95 Set annotation data (chip hgu95av2)*. R package version 3.2.3. 2016.

[28] Marc Carlson. *org.Hs.eg.db: Genome wide annotation for Human*. R package version 3.7.0. 2018.

[29] Marc Carlson. *org.Mm.eg.db: Genome wide annotation for Mouse*. R package version 3.7.0. 2018.

[30] Benilton Carvalho y Rafael Irizarry. *oligo: Preprocessing tools for oligonucleotide arrays*. R package version 1.46.0. 2018.

[31] Min Chen y col. «A powerful Bayesian meta-analysis method to integrate multiple gene set enrichment studies». En: *Bioinformatics* 29.7 (2013), págs. 862-869. DOI: 10.1093/bioinformatics/btt068. eprint: http://bioinformatics.oxfordjournals.org/content/29/7/862.full.pdf+html. URL: http://bioinformatics.oxfordjournals.org/content/29/7/862.abstract.

[32] Quan Chen, Xianghong J. Zhou y Fengzhu Sun. «Finding Genetic Overlaps Among Diseases Based on Ranked Gene Lists». En: *Journal of Computational Biology* 22.2 (feb. de 2015), págs. 111-123. ISSN: 1066-5277. DOI: 10.1089/cmb.2014.0149. URL: http://dx.doi.org/10.1089/cmb.2014.0149.

[33] Yunshun Chen y col. *edgeR: Empirical Analysis of Digital Gene Expression Data in R*. R package version 3.24.2. 2018. URL: http://bioinf.wehi.edu.au/edgeR.

[34] Peter J A Cock y col. «The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants». En: *Nucleic Acids Research* 38.6 (nov. de 2009), págs. 1767-1771. ISSN: 1362-4962. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2847217/.

[35] Ana Conesa y col. «A survey of best practices for RNA-seq data analysis». En: *Genome Biology* 17.1 (2016), págs. 1-19. ISSN: 1474-760X. DOI: 10.1186/s13059-016-0881-8. URL: http://dx.doi.org/10.1186/s13059-016-0881-8.

[36] Aedín C. Culhane y col. «GeneSigDB: a manually curated database and resource for analysis of gene expression signatures». En: *Nucleic Acids Research* 40.D1 (2012), págs. D1060-D1066. DOI: 10.1093/nar/gkr901. eprint: http://nar.oxfordjournals.org/content/40/D1/D1060.full.pdf+html. URL: http://nar.oxfordjournals.org/content/40/D1/D1060.abstract.

[37] Sean Davis. *GEOquery: Get data from NCBI Gene Expression Omnibus (GEO)*. R package version 2.50.5. 2018. URL: https://github.com/seandavi/GEOquery.

[38] Karl E. Peace Ding-Geng (Din) Chen. *Applied Meta-Analysis with R*. CRC Press, 3 de mayo de 2013. 342 págs. ISBN: 978-1-4665-0600-8. URL: http://www.ebook.de/de/product/21182622/ding_geng_din_chen_karl_e_peace_applied_meta_analysis_with_r.html.

[39] S. Dudoit, J.P. Shaffer y J.C. Boldrick. «Multiple hypothesis testing in microarray experiments». En: *Statistical Science* 18 (2003). Microarrays, págs. 71-103.

[40] Sandrine Dudoit y Robert Gentleman. *Cluster Analysis in DNA Microarray Experiments*. Bioconductor Short Course Winter 2002. Bioconductor.org. 2002. URL: http://www.bioconductor.org/help/course-materials/2002/Seattle02/Cluster/cluster.pdf.

[41] Steffen Durinck y Wolfgang Huber. *biomaRt: Interface to BioMart databases (i.e. Ensembl)*. R package version 2.38.0. 2018.

[42] Brad Efron y R. Tibshirani. *GSA: Gene set analysis*. R package version 1.03. 2010. URL: https://CRAN.R-project.org/package=GSA.

[43] Bradley Efron y Robert Tibshirani. «On testing the significance of sets». En: *Annals of Applied Statistics* 1.1 (2007). Gene set analysis, págs. 107-129. DOI: 10.1214/07-AOAS101.

[44] S. Falcon y R. Gentleman. «Using GOstats to test gene lists for GO term association». En: *Bioinformatics* 23.2 (2007), págs. 257-258. DOI: 10.1093/bioinformatics/btl567. eprint: http://bioinformatics.oxfordjournals.org/content/23/2/257.full.pdf+html. URL: http://bioinformatics.oxfordjournals.org/content/23/2/257.abstract.

[45] Julian Faraway. *faraway: Functions and Datasets for Books by Julian Faraway*. R package version 1.0.7. 2016. URL: https://CRAN.R-project.org/package=faraway.

[46] Kyle Furge y Karl Dykema. *PGSEA: Parametric Gene Set Enrichment Analysis*. R package version 1.56.0. 2018.

[47] Zhuohui Gan y col. «MAAMD: a workflow to standardize meta-analyses and comparison of affymetrix microarray data». En: *BMC Bioinformatics* 15.1 (2014), pág. 69. ISSN: 1471-2105. DOI: 10.1186/1471-2105-15-69. URL: http://www.biomedcentral.com/1471-2105/15/69.

[48] Laurent Gautier y col. «affy—analysis of Affymetrix GeneChip data at the probe level». En: *Bioinformatics* 20.3 (2004), págs. 307-315. ISSN: 1367-4803. DOI: http://dx.doi.org/10.1093/bioinformatics/btg405.

[49] Ludwig Geistlinger y col. *EnrichmentBrowser: Seamless navigation through combined results of set-based and network-based enrichment analysis*. R package version 2.12.0. 2018.

[50] Ludwig Geistlinger y col. «From sets to graphs: towards a realistic enrichment analysis of transcriptomic systems». En: *Bioinformatics* 27.13 (2011), págs. i366-i373. DOI: 10.1093/bioinformatics/btr228. eprint: http://bioinformatics.oxfordjournals.org/content/27/13/i366.full.pdf+html. URL: http://bioinformatics.oxfordjournals.org/content/27/13/i366.abstract.

[51] R. Gentleman. *annotate: Annotation for microarrays*. R package version 1.60.0. 2018.

[52] R. Gentleman y col. *Biobase: Base functions for Bioconductor*. R package version 2.42.0. 2018.

[53] R. Gentleman y col., eds. *Bioinformatics and Computational Biology Solutions Using R and Bioconductor*. Springer, 2005.

[54] R. Gentleman y col. *genefilter: methods for filtering genes from high-throughput experiments*. R package version 1.64.0. 2018.

[55] Robert Gentleman. *Category: Category Analysis*. R package version 2.48.0. 2018.

[56] Robert Gentleman. *GOstats: Tools for manipulating GO and microarrays*. R package version 2.48.0. 2018.

[57] Robert Gentleman. «Reproducible Research: A Bioinformatics Case Study». En: *Statistical Applications in Genetics and Molecular Biology* 4.1 (2005), pág. 2. DOI: 10.2202/1544-6115.1034.

[58] Robert Gentleman y Duncan Temple Lang. *Statistical Analyses and Reproducible Research*. Inf. téc. Bioconductor Project. Bioconductor Project Working Papers, 2004.

[59] J. J. Goeman y P. Buhlmann. «Analyzing gene expression data in terms of gene sets: methodological issues». En: *Bioinformatics* 23.8 (2007), págs. 980-987. DOI: 10.1093/bioinformatics/btm051. eprint: http://bioinformatics.oxfordjournals.org/content/23/8/980.full.pdf+html. URL: http://dx.doi.org/10.1093/bioinformatics/btm051.

[60] T. R. Golub y col. «Molecular classification of cancer: class discovery and class prediction by gene expression monitoring.» eng. En: *Science* 286.5439 (1999), págs. 531-537. URL: 10.1126/science.286.5439.53.

[61] Todd Golub. *golubEsets: exprSets for golub leukemia data*. R package version 1.24.0. 2018.

[62] Cedric Gondro. *Primer to Analysis of Genomic Data Using R*. Springer International Publishing, 2015. DOI: 10.1007/978-3-319-14475-7.

[63] Manuel Ugidos Guerrero. «Estudio del perfil de expresión de micro-RNA en pacientes de cáncer de mama». Tesis de mtría. Universidad de Valencia, 2017.

[64] Justin Guinney y Robert Castelo. *GSVA: Gene Set Variation Analysis for microarray and RNA-seq data*. R package version 1.30.0. 2018. URL: https://github.com/rcastelo/GSVA.

[65] Florian Hahne y col. *Bioconductor Case Studies*. Use R! Springer, 2008.

[66] Sonja Hanzelmann, Robert Castelo y Justin Guinney. «GSVA: gene set variation analysis for microarray and RNA-Seq data». En: *BMC Bioinformatics* 14.1 (2013), pág. 7. ISSN: 1471-2105. DOI: 10.1186/1471-2105-14-7. URL: http://www.biomedcentral.com/1471-2105/14/7.

[67] Masaaki Horikoshi y Yuan Tang. *ggfortify: Data Visualization Tools for Statistical Analysis Results*. R package version 0.4.5. 2018. URL: https://CRAN.R-project.org/package=ggfortify.

[68] R.A. Irizarry y col. «Exploration, normalization, and summaries of high density oligonucleotide array probe level data». En: *Biostatistics* 4.2 (2003), págs. 249-264. DOI: 10.1093/biostatistics/4.2.249. eprint: http://biostatistics.oxfordjournals.org/content/4/2/249.full.pdf+html. URL: http://biostatistics.oxfordjournals.org/content/4/2/249.abstract.

[69] R.A. Irizarry y col. «Gene set enrichment analysis made simple». En: *Statistical Methods in Medical Research* 18.6 (2009). Gene set analysis, págs. 565-575.

[70] Rafael A. Irizarry y col. *affy: Methods for Affymetrix Oligonucleotide Arrays*. R package version 1.60.0. 2018.

[71] Laurent Jacob, Pierre Neuvial y Sandrine Dudoit. *DEGraph: Two-sample tests on a graph*. R package version 1.34.0. 2018.

[72] Laurent Jacob, Pierre Neuvial y Sandrine Dudoit. «More power via graph-structured tests for differential expression of gene networks». En: *Ann. Appl. Stat.* 6.2 (jun. de 2012), págs. 561-600. DOI: 10.1214/11-AOAS528. URL: http://dx.doi.org/10.1214/11-AOAS528.

[73] Arnoud J. Kal y col. «Dynamics of Gene Expression Revealed by Comparison of Serial Analysis of Gene Expression Transcript Profiles from Yeast Grown on Two Different Carbon Sources». En: *Molecular Biology of the Cell* 10.6 (1999), págs. 1859-1872. DOI: 10.1091/mbc.10.6.1859. eprint: http://www.molbiolcell.org/content/10/6/1859.full.pdf+html. URL: http://www.molbiolcell.org/content/10/6/1859.abstract.

[74] Audrey Kauffmann, Ibrahim Emam y Michael Schubert. *ArrayExpress: Access the ArrayExpress Microarray Database at EBI and build Bioconductor data structures: ExpressionSet, AffyBatch, NChannelSet*. R package version 1.42.0. 2018.

[75] Audrey Kauffmann y Wolfgang Huber. *arrayQualityMetrics: Quality metrics report for microarray data sets*. R package version 3.38.0. 2018.

[76] L. Kaufman y P.J. Rousseeuw. *Finding Groups in Data. An Introduction to Cluster Analysis*. Wiley, 1990.

[77] Eija Korpelainen y col. *RNA-seq Data Analysis A Practical Approach*. CRC Press, 2015.

[78] Marcin Kosinski y Przemyslaw Biecek. *RTCGA: The Cancer Genome Atlas Data Integration*. R package version 1.12.0. 2018. URL: https://rtcga.github.io/RTCGA.

[79] Jun Li y Robert Tibshirani. «Finding consistent patterns: A nonparametric approach for identifying differential expression in RNA-Seq data». En: *Statistical Methods in Medical Research* 22.5 (2013), págs. 519-536. DOI: 10.1177/0962280211428386. eprint: http://smm.sagepub.com/content/22/5/519.full.pdf+html. URL: http://smm.sagepub.com/content/22/5/519.abstract.

[80] Xiaochun Li. *ALL: A data package*. R package version 1.24.0. 2018.

[81] Q. Liu y col. «Comparative evaluation of gene-set analysis methods». En: *BMC Bioinformatics* 8.1 (2007), págs. 1-15. ISSN: 1471-2105. DOI: 10.1186/1471-2105-8-431. URL: http://dx.doi.org/10.1186/1471-2105-8-431.

[82] Michael Love. *parathyroidSE: RangedSummarizedExperiment for RNA-Seq of primary cultures of parathyroid tumors by Haglund et al., J Clin Endocrinol Metab 2012.* R package version 1.20.0. 2018.

[83] Michael Love, Wolfgang Huber y Simon Anders. «Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2». En: *Genome Biology* 15.12 (2014), pág. 550. ISSN: 1465-6906. DOI: 10.1186/s13059-014-0550-8. URL: http://genomebiology.com/2014/15/12/550.

[84] P. Lund Steven y col. *Detecting Differential Expression in RNA-sequence Data Using Quasi-likelihood with Shrunken Dispersion Estimates*. 2012. URL: //www.degruyter.com/view/j/sagmb.2012.11.issue-5/1544-6115.1826/1544-6115.1826.xml.

[85] Steve Lund y col. *QuasiSeq: Analyzing RNA Sequencing Count Tables Using Quasi-Likelihood*. R package version 1.0-10-2. 2018. URL: https://CRAN.R-project.org/package=QuasiSeq.

[86] James W. MacDonald. *affycoretools: Functions useful for those doing repetitive analyses with Affymetrix GeneChips*. R package version 1.54.0. 2018.

[87] Henryk Maciejewski. «Gene set analysis methods: statistical models and methodological differences». En: *Briefings in Bioinformatics* (2013). DOI: 10.1093/bib/bbt002. eprint: http://bib.oxfordjournals.org/content/early/2013/02/09/bib.bbt002.full.pdf+html. URL: http://bib.oxfordjournals.org/content/early/2013/02/09/bib.bbt002.abstract.

[88] S. C. Madeira y A. L. Oliveira. «Biclustering algorithms for biological data analysis: a survey». En: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 1.1 (2004), págs. 24-45. DOI: 10.1109/TCBB.2004.2.

[89] Martin Maechler y col. *cluster: "Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.* R package version 2.0.7-1. 2018. URL: https://CRAN.R-project.org/package=cluster.

[90] D. Maglott y col. «Entrez Gene: gene-centered information at NCBI». En: *Nucleic Acids Research* 39.Database (nov. de 2010), págs. D52-D57. DOI: 10.1093/nar/gkq1237.

[91] Paolo Martini y col. «Along signal paths: an empirical gene set approach exploiting pathway topology». En: *Nucleic Acids Research* 41.1 (2013), e19. DOI: 10.1093/nar/gks866. eprint: http://nar.oxfordjournals.org/content/41/1/e19.full.pdf+html. URL: http://nar.oxfordjournals.org/content/41/1/e19.abstract.

[92] Qinxue Meng y col. «DBNorm: normalizing high-density oligonucleotide microarray data based on distributions». En: *BMC Bioinformatics* 18.1 (nov. de 2017). DOI: 10.1186/s12859-017-1912-5.

[93] V.K . Mootha y col. «PGC-1alpha-responsive genes involved in oxidative phosphorylation are coordinately downregulated in human diabetes». En: *Nature Genetics* 34 (2003), págs. 267-73.

[94] Martin Morgan, Seth Falcon y Robert Gentleman. *GSEABase: Gene set enrichment data structures and methods*. R package version 1.44.0. 2018.

[95] Martin Morgan, Michael Lawrence y Simon Anders. *ShortRead: FASTQ input and manipulation*. R package version 1.40.0. 2018.

[96] Martin Morgan y col. *Rsamtools: Binary alignment (BAM), FASTA, variant call (BCF), and tabix file import*. R package version 1.34.0. 2018. URL: http://bioconductor.org/packages/release/bioc/html/Rsamtools.html.

[97] Martin Morgan y col. *SummarizedExperiment: SummarizedExperiment container*. R package version 1.12.0. 2018.

[98] Ali Mortazavi y col. «Mapping and quantifying mammalian transcriptomes by RNA-Seq». En: *Nat Meth* 5.7 (jul. de 2008), págs. 621-628. ISSN: 1548-7091. URL: http://dx.doi.org/10.1038/nmeth.1226.

[99] Haroon Naeem y col. «Rigorous assessment of gene set enrichment tests». En: *Bioinformatics* 28.11 (2012), págs. 1480-1486. DOI: 10.1093/bioinformatics/bts164. eprint: http://bioinformatics.oxfordjournals.org/content/28/11/1480.full.pdf+html. URL: http://bioinformatics.oxfordjournals.org/content/28/11/1480.abstract.

[100] Richard M. Neve y col. «A collection of breast cancer cell lines for the study of functionally distinct cancer subtypes». En: *Cancer cell* 10.6 (2006), págs. 515-527. DOI: 10.1016/j.ccr.2006.10.008.

[101] Michael A. Newton y col. «Random-set methods identify distinct aspects of the enrichment signal in gene-set analysis». En: *Annals of Appied Statistics* 1.1 (2007), págs. 85-106. DOI: 10.1214/07-AOAS104. URL: http://projecteuclid.org/euclid.aoas/1183143730.

[102] Tobias Oetiker y col. *La introducción no-tan-corta a LATeX 2e*. 2010.

[103] Alicia Oshlack, Mark Robinson y Matthew Young. «From RNA-seq reads to differential expression results». En: *Genome Biology* 11.12 (2010), pág. 220. ISSN: 1465-6906. DOI: 10.1186/gb-2010-11-12-220. URL: http://genomebiology.com/2010/11/12/220.

[104] Gabriel Östlund y Erik L.L. Sonnhammer. «Avoiding pitfalls in gene (co)expression meta-analysis». En: *Genomics* 103.1 (2014), págs. 21-30. ISSN: 0888-7543. DOI: http://dx.doi.org/10.1016/j.ygeno.2013.10.006. URL: http://www.sciencedirect.com/science/article/pii/S0888754313002012.

[105] Hervé Pagès y col. *AnnotationDbi: Annotation Database Interface*. R package version 1.44.0. 2018.

[106] J. K. Pickrell y col. «Understanding mechanisms underlying human gene expression variation with RNA sequencing». En: *Nature* 464 (2010). DOI: 10.1038/nature08872. URL: http://dx.doi.org/10.1038/nature08872.

[107] Terri D. Pigott. *Advances in Meta-Analysis*. Springer US, 2012. DOI: 10.1007/978-1-4614-2278-5.

[108] Katherine S. Pollard y col. *multtest: Resampling-based multiple hypothesis testing*. R package version 2.38.0. 2018.

[109] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2018. URL: https://www.R-project.org/.

[110] Yasir Rahmatallah, Frank Emmert-Streib y Galina Glazko. «Comparative evaluation of gene set analysis approaches for RNA-Seq data». En: *BMC Bioinformatics* 15.1 (2014), págs. 1-15. ISSN: 1471-2105. DOI: 10.1186/s12859-014-0397-8. URL: http://dx.doi.org/10.1186/s12859-014-0397-8.

[111] C.R. Rao. *Linear Statistical Inference and Its Applications*. Wiley, 1967.

[112] Anat Reiner, Daniel Yekutieli y Yoav Benjamini. «Identifying differentially expressed genes using false discovery rate controlling procedures». En: *Bioinformatics* 19.3 (2003), págs. 368-375. DOI: 10.1093/bioinformatics/btf877. eprint: http://bioinformatics.oxfordjournals.org/content/19/3/368.full.pdf+html. URL: http://bioinformatics.oxfordjournals.org/content/19/3/368.abstract.

[113] Tyler Rinker y Dason Kurkiewicz. *pacman: Package Management Tool*. R package version 0.5.0. 2018. URL: https://CRAN.R-project.org/package=pacman.

[114] David Robinson y Alex Hayes. *broom: Convert Statistical Analysis Objects into Tidy Tibbles*. R package version 0.5.1. 2018. URL: https://CRAN.R-project.org/package=broom.

[115] M. D. Robinson y A. Oshlack. «A scaling normalization method for differential expression analysis of RNA-seq data». En: *Genome Biol* 11.3 (2010), R25. ISSN: 1465-6906. DOI: 10.1186/gb-2010-11-3-r25. URL: http://dx.doi.org/10.1186/gb-2010-11-3-r25.

[116] Mark D. Robinson y Gordon K. Smyth. «Moderated statistical tests for assessing differences in tag abundance». En: *Bioinformatics* 23.21 (2007), págs. 2881-2887. DOI: 10.1093/bioinformatics/btm453. eprint: http://bioinformatics.oxfordjournals.org/content/23/21/2881.full.pdf+html. URL: http://bioinformatics.oxfordjournals.org/content/23/21/2881.abstract.

[117] Mark D. Robinson y Gordon K. Smyth. «Small-sample estimation of negative binomial dispersion, with applications to SAGE data». En: *Biostatistics* 9.2 (2008), págs. 321-332. DOI: 10.1093/biostatistics/kxm030. eprint: http://biostatistics.oxfordjournals.org/content/9/2/321.full.pdf+html. URL: http://biostatistics.oxfordjournals.org/content/9/2/321.abstract.

[118] Gabriele Sales y col. «Graphite Web: web tool for gene set analysis exploiting pathway topology». En: *Nucleic Acids Research* (2013). DOI: 10.1093/nar/gkt386. eprint: http://nar.oxfordjournals.org/content/early/2013/05/10/nar.gkt386.full.pdf+html. URL: http://nar.oxfordjournals.org/content/early/2013/05/10/nar.gkt386.abstract.

[119] Guido Schwarzer, James R. Carpenter y Gerta Rücker. *Meta-Analysis with R*. Springer International Publishing, 2015. DOI: 10.1007/978-3-319-21416-0.

[120] Fatemeh Seyednasrollah, Asta Laiho y Laura L. Elo. «Comparison of software packages for detecting differential expression in RNA-seq studies». En: *Briefings in Bioinformatics* (2013). DOI: 10.1093/bib/bbt086. eprint: http://bib.oxfordjournals.org/content/early/2013/12/02/bib.bbt086.full.pdf+html. URL: http://bib.oxfordjournals.org/content/early/2013/12/02/bib.bbt086.abstract.

[121] Mirko Signorelli, Veronica Vinciotti y Ernst C. Wit. «NEAT: an efficient network enrichment analysis test». En: *BMC Bioinformatics* 17.1 (2016), págs. 1-17. ISSN: 1471-2105. DOI: 10.1186/s12859-016-1203-6. URL: http://dx.doi.org/10.1186/s12859-016-1203-6.

[122] Mirko Signorelli, Veronica Vinciotti y Ernst C. Wit. *neat: Efficient Network Enrichment Analysis Test*. R package version 1.1.3. 2018. URL: https://CRAN.R-project.org/package=neat.

[123] P.P. Sinha. *Bioinformatics with R Cookbook*. Packt Publishing, 2014.

[124] Martin Slawski y Anne-Laure Boulesteix. *GeneSelector: Stability and Aggregation of ranked gene lists*. R package version 2.32.0. 2018.

[125] Gordon K. Smyth. «Linear Models and Empirical Bayes Methods for Assessing Differential Expression in Microarray Experiments». En: *Statistical Applications in Genetics and Molecular Biology* 1 (2004), pág. 3.

[126] Gordon Smyth y col. *limma: Linear Models for Microarray Data*. R package version 3.38.3. 2018. URL: http://bioinf.wehi.edu.au/limma.

[127] Peter H.A. Sneath y Rober R. Sokal. *Numerical Taxonomy. The principes and practice of numerical Classification*. San Francisco, USA: W.H. Freeman y Company, 1973.

[128] Phillip Stafford, ed. *Methods in Microarray Normalization*. CRC Press, 2008.

[129] John D. Storey y Robert Tibshirani. «Statistical significance for genomewide studies». En: *Proceedings of the National Academy of Sciences* 100.16 (2003), págs. 9440-9445. DOI: 10.1073/pnas.1530509100. eprint: http://www.pnas.org/content/100/16/9440.full.pdf+html. URL: http://www.pnas.org/content/100/16/9440.abstract.

[130] Aravind Subramanian y col. «Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles». En: *Proceedings of the National Academy of Sciences of the United States of America* 102.43 (2005), págs. 15545-15550. DOI: 10.1073/pnas.0506580102. eprint: http://www.pnas.org/content/102/43/15545.full.pdf+html. URL: http://www.pnas.org/content/102/43/15545.abstract.

[131] Dan Tenenbaum. *KEGGREST: Client-side REST access to KEGG*. R package version 1.22.0. 2018.

[132] Lu Tian y col. «Discovering statistically significant pathways in expression profiling studies». En: *Proceedings of the National Academy of Sciences of the United States of America* 102.38 (2005), págs. 13544-13549. DOI: 10.1073/pnas.0506577102. eprint: http://www.pnas.org/content/102/38/13544.full.pdf+html. URL: http://www.pnas.org/content/102/38/13544.abstract.

[133] R. Tibshirani y col. *samr: SAM: Significance Analysis of Microarrays*. R package version 3.0. 2018. URL: https://CRAN.R-project.org/package=samr.

[134] Shailesh Tripathi, Galina V. Glazko y Frank Emmert-Streib. «Ensuring the statistical soundness of competitive gene set approaches: gene filtering and genome-scale coverage are essential». En: *Nucleic Acids Research* 41.7 (2013), e82. DOI: 10.1093/nar/gkt054. eprint: http://nar.oxfordjournals.org/content/41/7/e82.full.pdf+html. URL: http://nar.oxfordjournals.org/content/41/7/e82.abstract.

[135] Olga Troyanskaya y col. «Missing value estimation methods for DNA microarrays». En: *Bioinformatics* 17.6 (2001), págs. 520-525. DOI: 10.1093/bioinformatics/17.6.520. eprint: http://bioinformatics.oxfordjournals.org/content/17/6/520.full.pdf+html. URL: http://bioinformatics.oxfordjournals.org/content/17/6/520.abstract.

[136] Virginia Goss Tusher, Robert Tibshirani y Gilbert Chu. «Significance analysis of microarrays applied to the ionizing radiation response». En: *Proceedings of the National Academy of Sciences* 98.9 (2001). Gene set analysis, págs. 5116-5121. DOI: 10.1073/pnas.091062498. eprint: http://www.pnas.org/content/98/9/5116.full.pdf+html. URL: http://www.pnas.org/content/98/9/5116.abstract.

[137] Marie Verbanck, Sebastien Le y Jerome Pages. «A new unsupervised gene clustering algorithm based on the integration of biological knowledge into expression data». En: *BMC Bioinformatics* 14.1 (2013), pág. 42. ISSN: 1471-2105. DOI: 10.1186/1471-2105-14-42. URL: http://www.biomedcentral.com/1471-2105/14/42.

[138] John Verzani. *UsingR: Data Sets, Etc. for the Text "Using R for Introductory Statistics", Second Edition*. R package version 2.0-6. 2018. URL: https://CRAN.R-project.org/package=UsingR.

[139] Zhong Wang, Mark Gerstein y Michael Snyder. «RNA-Seq: a revolutionary tool for transcriptomics». En: *Nat Rev Genet* 10.1 (ene. de 2009), págs. 57-63. ISSN: 1471-0056. URL: http://dx.doi.org/10.1038/nrg2484.

[140] Hadley Wickham. *R Packages: Organize, Test, Document, and Share Your Code*. O'Reilly Media, 2015. ISBN: 978-1-49191-059-7. URL: http://gen.lib.rus.ec/book/index.php?md5=C0B7AD084B65C02500D0029AA4B218C9.

[141] Hadley Wickham. *reshape: Flexibly Reshape Data*. R package version 0.8.8. 2018. URL: https://CRAN.R-project.org/package=reshape.

[142] Hadley Wickham. «Reshaping Data with the reshape Package». En: *Journal of Statistical Software* 21.1 (2007), págs. 1-20. ISSN: 1548-7660. DOI: 10.18637/jss.v021.i12. URL: https://www.jstatsoft.org/index.php/jss/article/view/v021i12.

[143] Hadley Wickham. «Tidy Data». En: *Journal of Statistical Software* 59.1 (2014), págs. 1-23. ISSN: 1548-7660. DOI: 10.18637/jss.v059.i10. URL: https://www.jstatsoft.org/index.php/jss/article/view/v059i10.

[144] Hadley Wickham, Peter Danenberg y Manuel Eugster. *roxygen2: In-Line Documentation for R*. R package version 6.1.1. 2018. URL: https://CRAN.R-project.org/package=roxygen2.

[145] Hadley Wickham y col. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.1.0. 2018. URL: https://CRAN.R-project.org/package=ggplot2.

[146] E. Wit y J.D. McClure. *Statistics for microarrays: design, analysis, and inference*. Wiley, 2004.

[147] Daniela M Witten y Robert Tibshirani. «Scientific research in the age of omics: the good, the bad, and the sloppy». En: *Journal of the American Medical Informatics Association* 20.1 (2013), págs. 125-127. DOI: 10.1136/amiajnl-2012-000972. eprint: http://jamia.bmj.com/content/20/1/125.full.pdf+html. URL: http://jamia.bmj.com/content/20/1/125.abstract.

[148] Di Wu y Gordon K. Smyth. «Camera: a competitive gene set test accounting for inter-gene correlation». En: *Nucleic Acids Research* 40.17 (2012), e133. DOI: 10.1093/nar/gks461. eprint: http://nar.oxfordjournals.org/content/40/17/e133.full.pdf+html. URL: http://nar.oxfordjournals.org/content/40/17/e133.abstract.

[149] Jean Wu y Rafael Irizarry with contributions from James MacDonald Jeff Gentry. *gcrma: Background Adjustment Using Sequence Information*. R package version 2.54.0. 2018.

[150] Jianguo Xia, Erin E Gill y Robert E W Hancock. «NetworkAnalyst for statistical, visual and network-based meta-analysis of gene expression data». En: *Nat. Protocols* 10.6 (jun. de 2015), págs. 823-844. ISSN: 1754-2189. URL: http://dx.doi.org/10.1038/nprot.2015.052.

[151] Yihui Xie. *Dynamic Documents with R and knitr*. 2nd. Chapman & Hall/CRC The R Series. Chapman y Hall/CRC, 2015.

[152] Yihui Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.21. 2018. URL: https://CRAN.R-project.org/package=knitr.

[153] Qing Xiong y col. «Integrating genetic and gene expression evidence into genome-wide association analysis of gene sets». En: *Genome Research* 22.2 (2012), págs. 386-397. DOI: 10.1101/gr.124370.111. eprint: http://genome.cshlp.org/content/22/2/386.full.pdf+html. URL: http://genome.cshlp.org/content/22/2/386.abstract.

[154] Matthew Young. *geneLenDataBase: Lengths of mRNA transcripts for a number of genomes*. R package version 1.18.0. 2018.

[155] Yi-Hui Zhou, William T. Barry y Fred A. Wright. «Empirical pathway analysis, without permutation». En: *Biostatistics* (2013). DOI: 10.1093/biostatistics/kxt004. eprint: http://biostatistics.oxfordjournals.org/content/early/2013/02/20/biostatistics.kxt004.full.pdf+html. URL: http://biostatistics.oxfordjournals.org/content/early/2013/02/20/biostatistics.kxt004.abstract.

[156] Jack Zhu y Sean Davis. *SRAdb: A compilation of metadata from NCBI SRA and tools*. R package version 1.44.0. 2018. URL: http://gbnci.abcc.ncifcrf.gov/sra/.

Índice alfabético

Glosario

Affymetrix http://www.affymetrix.com/. 35

Agilent https://www.agilent.com/.. 105

BAM Es la versión binaria del formato SAM. http://genome.sph.umich.edu/wiki/SAM y la especificación del formato la tenemos en http://samtools.sourceforge.net/SAM1.pdf.. 117

Bioconductor Red de espejos con paquetes de R para análisis de datos ómicos: https://www.bioconductor.org/. 28, 340

BioMart http://www.biomart.org Es un sistema de almacenamiento de datos orientado a las consultas. Ha sido desarrollado por el European Bioinformatics Institute (EBI) y el Cold Spring Harbor Laboratory (CSHL). . 53

Bowtie2 Alineador de secuencias. http://bowtie-bio.sourceforge.net/bowtie2/index.shtml . 119, 120, 131

CDS Es la región codificante de un gen: **C**oding **D**NA **S**equence. Es la parte del gen (DNA o RNA) compuesta por los exones que codifican proteína. El CDS es la porción de un transcrito que es trasladado por un ribosoma. . 43, 46

CRAN Red de espejos con paquetes de R: https://cran.r-project.org/ . 28, 340

Ensembl https://en.wikipedia.org/wiki/Ensembl. 37, 38, 54, 56, 57

Entrez https://en.wikipedia.org/wiki/Entrez. 35, 37, 54, 56, 57

ENTREZID [90].. 105

ExpressionSet Clase S4 para almacenar datos de expresión de microarrays. Definida en [52].. xv, 24, 215

FASTQ Es un formato para guardar datos de secuencias. Consiste de cuatro líneas. La primera contiene el nombre de la secuencia. La segunda línea contiene a la propia secuencia. La tercera línez contiene información opcional sobre la secuencia. La cuarta línea cuantifica la confianza o calidad en la determinación de cada base recogida en la segunda línea. . 112, 117

Gene Ontology Abreviadamente GO. https://en.wikipedia.org/wiki/Gene_ontology y en http://geneontology.org/.. 37, 38, 57, 240, 245, 252, 258, 259, 265, 266, 277, 286

GEO NCBI GEO (Gene Expression Omnibus) es una base de datos de datos de expresión obtenidos con microarrays. También tiene algo de datos de secuenciación. Su dirección es http://www.ncbi.nlm.nih.gov/geo/. . 62, 99, 102, 103, 111, 123

GitHub Es un lugar web para desarrollo colaborativo de software. http://bowtie-bio.sourceforge.net/bowtie2/index.shtml . 369

GTEx Herramienta de análisis en línea que incluye análisis transcriptómico y estudios de asociación genética. Los datos son propios de la herramienta. https://www.gtexportal.org/home/.. 5

GTF/GFF Formato de fichero cuya descripción detallada se puede encontrar en http://www.ensembl.org/info/website/upload/gff.html.. 118

KEGG http://www.genome.jp/kegg/ https://en.wikipedia.org/wiki/KEGG Es una colección de bases de datos sobre genomas, rutas biológicas, enfermedades, medicamentos y sustancias químicas. . 245, 252, 286

OrgDb Tipo de paquetes de anotación centrados en el organismos a que se refieren. Por ejemplo [28, org.Hs.eg.db].. 340

R R es un entorno de programación estadístico. https://www.r-project.org/ . xv, 85, 92, 111, 293, 345, 347

RNA-Seq https://en.wikipedia.org/wiki/RNA-Seq. . 109

RPKM Reads per kilobase per million reads. C es el total de lecturas alineadas sobre la característica. El total de lecturas que podemos alinear es N (o tamaño de la librería), L la longitud de la característica de interés en bp. $RPKM = \frac{10^9 C}{NL}$. . 127

SAM Sequence Alignment/Map https://samtools.github.io/hts-specs/SAMv1.pdf . 118, 120

SAM http://genome.sph.umich.edu/wiki/SAM y la especificación del formato la tenemos en http://samtools.sourceforge.net/SAM1.pdf.. 117, 367

Samtools Es un conjunto de programas para trabajar con datos de secuenciación. Ver http://www.htslib.org/. En Debian/Ubuntu se instala con *a*pt-get install samtools. En R/Bioconductor tenemos el paquete [96, Rsamtools].. 117

SRA NCBI SRA (**S**equence **R**ead **A**rchive) es una base de datos con datos de secuenciación de DNA en forma de lecturas cortas generadas mediante secuenciación de alto rendimiento. Su dirección es http://www.ncbi.nlm.nih.gov/sra. En http://www.ncbi.nlm.nih.gov/books/NBK47528/ tenemos más información. Si no funciona el enlace buscamos en Google *SRA handbook*.. 111, 117

STAR Es un ejecutable que se baja desde GitHub `https://github.com/alexdobin/STAR/archive/master.zip`. Con *STAR aligner* en Google lo encontramos... 117

TCGA **T**he **C**ancer **G**enome **A**tlas es una base de datos online con estudios de cáncer utilizando distintas técnicas `https://cancergenome.nih.gov/`. . 5