Preliminares

M. Carme Ruíz de Villa
Alex Sánchez-Pla

PID_00192742

Universitat Oberta de Catalunya

Índice

1. Introducción a los microarrays

1.1. Antecedentes históricos

La biología molecular ha estado interesada desde sus comienzos en poder determinar el nivel de expresión de los genes integrados en el genoma humano. Para medir estos niveles dispone desde hace años de múltiples técnicas, tales como el `Northern blot` (a nivel de ARN) o el `Western blot` (a nivel de proteína).

Especial interés ha tenido siempre sobre las otras biomoléculas el estudio de los niveles del ARN transcrito. Hace unos años se acuñó el término de *transcriptómica* para referirse al estudio del ARN en cada una de sus formas. La transcriptómica ha contribuido a tener una mejor comprensión de la patogénesis de las enfermedades.

> En el año 2009, Wang y colaboradores ([13]) definieron el transcriptoma como el conjunto completo de tránscritos en una célula y su cantidad en un momento determinado del desarrollo o en una determinada condición fisiológica.

Como objetivos de la transcriptómica se han definido tres principales:

1) Catalogar todas las especies de tránscritos, incluyendo ARNm, ARN no codificante y pequeños ARNs.

2) Determinar la estructura transcripcional de los genes, en términos de sus puntos 5' de inicio y 3' de finalización, las modificaciones postranscripcionales y los patrones de *splicing*.

3) Cuantificar los diferentes niveles de expresión de cada tránscrito durante el desarrollo y bajo diferentes condiciones.

El interés de la transcriptómica no solamente se ha centrado en el desarrollo de nuevas tecnologías que mejoran su estudio, sino también en el desarrollo de nuevos métodos de extraer la gran cantidad de información que se genera con estas nuevas técnicas.

1.2. Los microarrays

Una de las técnicas que revolucionó el estudio del transcriptoma fueron los microarrays.

Un microarray es un artefacto para la realización de experimentos que permite estudiar simultáneamente múltiples unidades b, que representan los genes, proteínas o metabolitos, sobre un sustrato sólido de cristal, plástico o sílice, y expuestos a la acción de las moléculas diana cuya expresión se desea analizar (figura 1).

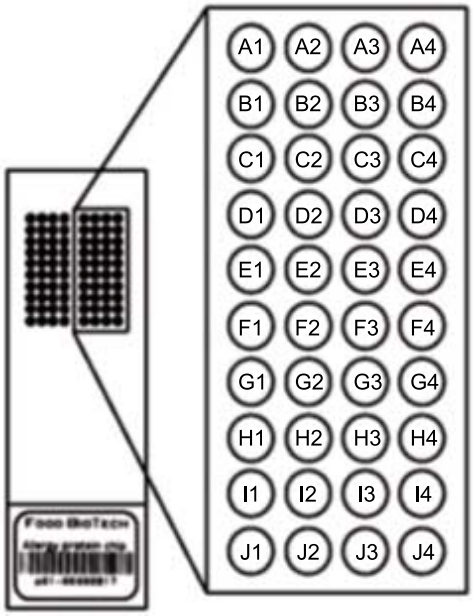

Figura 1. Representación esquemática de un microarray. Cada posición o *spot* contiene un gran número de copias de la secuencia completa o parcial de los genes o tránscritos cuya expresión se desea cuantificar.

La utilización de los microarrays en la última década ha generado inmensas cantidades de datos útiles para el estudio y el desarrollo de enfermedades, dando a conocer múltiples mapas de expresión génica, encontrar biomarcadores o construir firmas génicas para determinadas enfermedades.

Lo que caracteriza a los nuevos métodos utilizados para estudiar el transcriptoma no es lo que pueden medir, sino la cantidad de mediciones simultáneas que pueden realizar. Mientras que hasta hace apenas una década se estudiaban los genes uno a uno en profundidad, a partir del uso de estas nuevas tecnologías se pueden estudiar muchísimos genes a la vez, pero en contrapartida, con mucho menos detalle y más ruido.

Los microarrays, hoy una metodología bien consolidada, han sido cruciales para concebir una nueva manera de estudiar el transcriptoma, especialmente en el campo de la expresión génica. Después de ellos han venido otras técnicas que tienen en común con ellos el alto rendimiento, es decir, la capacidad para

medir muchas variables –cientos o miles– a la vez. Entre estas técnicas podemos destacar los arrays de SNPs, los microRNAs, la metilación y especialmente la secuenciación de nueva generación.

1.2.1. Aplicaciones de los microarrays

La tecnología de los microarrays se ha aplicado a una inmensa variedad de problemas, desde el estudio de enfermedades, como el cáncer o la esclerosis múltiple, al de los ritmos circadianos de las frutas. Entre otros temas los microarrays se han aplicado a:

- Estudio de genes que se expresan diferencialmente entre varias condiciones (sanos frente a enfermos, mutantes frente a salvajes, tratados frente a no tratados) ([2], [7], [9]).

- Clasificación molecular en enfermedades complejas ([6], [1]).

- Identificación de genes característicos de una patología (firma o *signature*) ([4]).

- Predicción de respuesta a un tratamiento ([10]).

- Y una gran variedad de otros temas.

1.3. Cómo funcionan los microarrays

En términos generales, los microarrays funcionan mediante la hibridación de una sonda específica (*probe*) y una molécula diana (*target*). La hibridación que ha tenido lugar se detecta mediante fluorescencia y se visualiza con la ayuda de un escáner. Los niveles de fluorescencia detectados reflejan la cantidad de moléculas diana presentes en la muestra problema.

Existen diferentes tipos de microarrays según la naturaleza de lo que se está estudiando. Podemos encontrar microarrays de proteínas, de tejidos, de ADN o de ARN (también llamados de expresión).

Una clasificación habitual de los microarrays es por el número de muestras que se hibridan simultáneamente. Distinguimos:

1) Microarrays de dos colores o *spotted arrays*.
2) Microarrays de un color o arrays de oligonucleótidos.

1.3.1. Microarrays de dos colores

Estos arrays aparecieron a mediados de los años noventa ([11]) y están basados en la hibridación competitiva de dos muestras, cada una de las cuales ha sido marcada con un marcador fluorescente diferente (normalmente Cy3 y Cy5, verde y rojo respectivamente).

En el array se imprimen las sondas (una sonda $\simeq$ un gen) cuyas secuencias se obtienen de información almacenada en bases de datos de secuencias como `GenBank`, `dbEST`, etc.

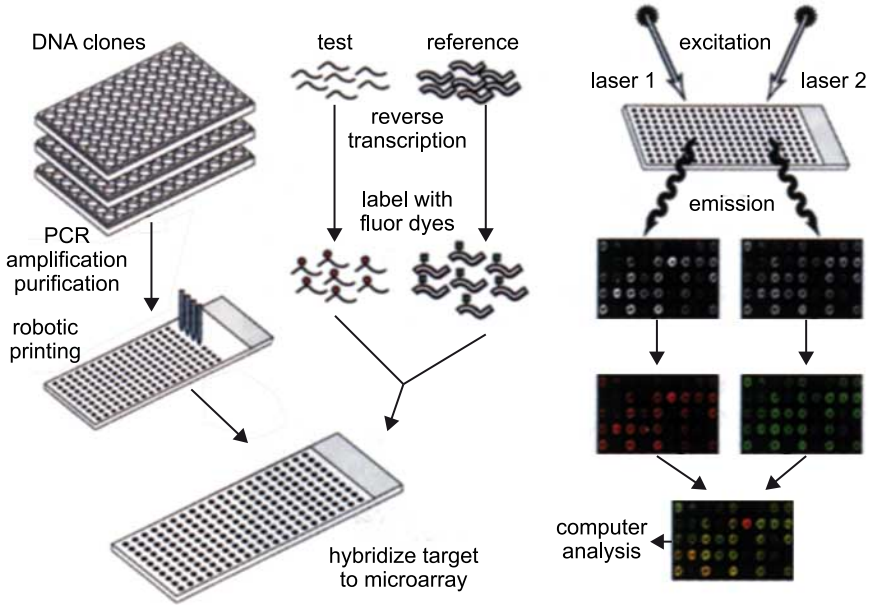

Figura 2. Esquema del funcionamiento de un microarrays de dos colores
En los microarrays de dos colores las sondas son sintetizadas *in vitro* y depositadas directamente sobre una superficie de cristal.

El ARN de las muestras problemas es extraído y posteriormente marcado con los marcadores fluorescentes, según a qué grupo a comparar pertenezcan. Las muestras marcadas se mezclan y se hibridan sobre el array. La hibridación consiste en combinar la muestra (*target*) y el microarray (sondas) y dejarlos un tiempo en una cámara de hibridación a una temperatura y agitación determinadas. Las sondas que tengan secuencias complementarias en las muestras se hibridarán con ellas y quedarán fuertemente adheridas. Pasadas unas horas se lava el microarray para eliminar los targets que no se hayan hibridado. Después de la hibridación, el array se ilumina con un láser que provoca que el marcador fluorescente emita fluorescencia de uno u otro color generando dos imágenes que se superpondrán para su análisis conjunto. La cantidad de fluorescencia generada es proporcional a la cantidad de ARNm presente en la muestra problema. El resultado final es un valor que representa el nivel de expresión de una muestra respecto a la otra, por lo que se le denomina expresión relativa.

1.3.2. Microarrays de un color

Como su propio nombre indica, en estos arrays las muestras están marcadas únicamente con un marcador fluorescente. En cada array solamente se hibrida una muestra, por lo que no se da la hibridación competitiva como pasaba en los arrays de dos colores. El valor que se obtiene después de iluminar el array con el láser es una medida numérica que se obtiene directamente del escáner, es decir, no está referida al valor de otra muestra, por lo que recibe el nombre de expresión absoluta.

La empresa Affymetrix (Santa Clara, California) es la casa comercial líder en la manufacturación y venta de este tipo de microarrays. Affymetrix sintetiza las sondas directamente sobre el chip mediante un proceso llamado fotolitografía. Este proceso consiste en la adición cíclica de los cuatro nucleótidos (adenina, timina, citosina y guanina) sobre la superficie rígida, donde existen ancladas unas especies químicas reactivas que se protegen y desprotegen para añadir el nucleótido deseado, mediante ciclos de luz y oscuridad. Así se consigue la síntesis de oligonucleótidos de unos 25-mer (mer = bases) de longitud. En la figura 3 se presenta el proceso de forma esquemática.

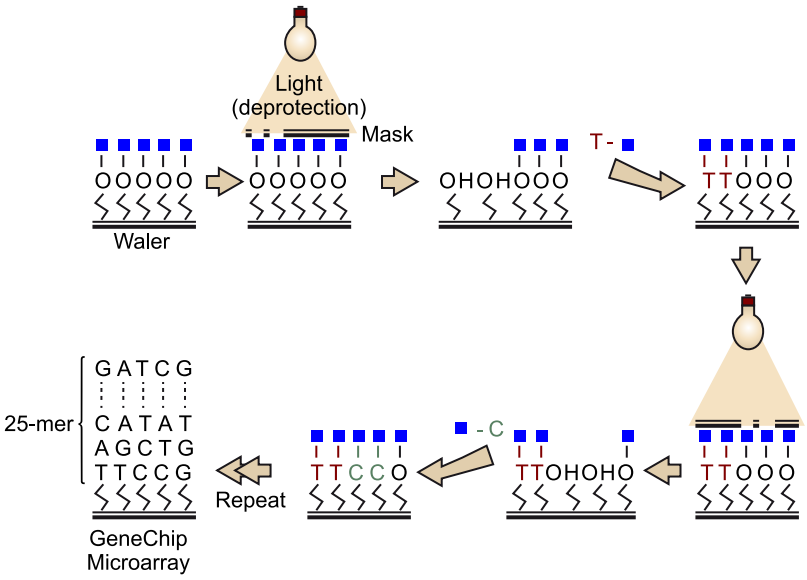

Figura 3. Esquema del proceso de fabricación de microarrays de oligonucleótidos

Cada oligonucleótido sintetizado de 25-mer (es decir, 25 bases) se denomina sonda (*probe*). Alrededor de 40x107 de estas sondas se agrupan en una celda llamada *probe cell*. Las probe cell están organizadas en parejas, probe pairs, donde se combinan un *Perfect Match* (PM) **-cuya secuencia de 25 mer coincide perfectamente con una parte del gen- y un Mismatch (MM)** -idéntico al PM, excepto en el nucleótido central, que no coincide (figura 4).

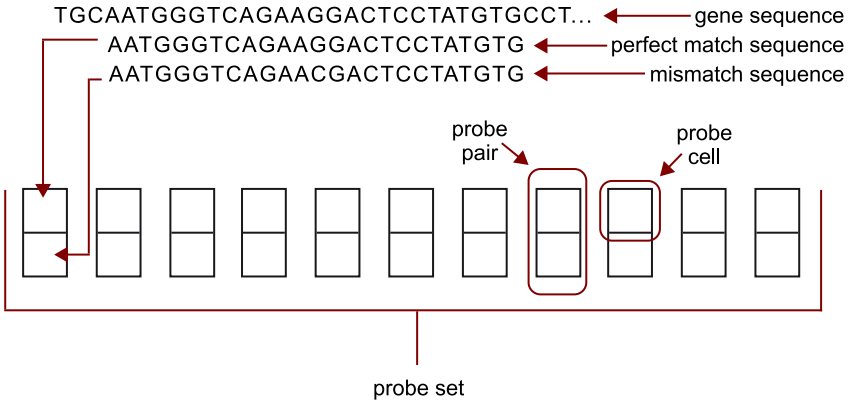

Figura 4. Estructura de un grupo de sondas (*probe sets* de un array tipo genechip de Affymetrix), en el que se muestra los distintos términos definidos: *probe-set*, *probe-pair*, *Perfect-Match*, *Mismatch*.

De 11 a 20 *probe pairs* se agrupan para formar un *probe set* o grupo de ondas. Diferentes *probe sets* se distribuyen a lo largo del array, y son las que definen a qué gen se unen y las que definen los niveles de expresión detectados.

El *Mismatch* (MM) se creó originalmente para que proporcionara una medida de la unión inespecífica (es decir, lo que se hibridara con esta secuencia parecida pero distinta a la original se consideraría ruido). Aunque algunos algoritmos originales de Affymetrix todavía lo utilizan, hoy en día ha caído en desuso, muchos algoritmos modernos como RMA –que se explicará más adelante– ya no lo utilizan y en las nuevas versiones de los arrays (modelos *GeneChip® Human Gene ST Arrays* de Affymetrix por ejemplo) ya no se incluyó.

1.3.3. Realización de un experimento de microarrays

Tanto en los arrays de un color como en los de dos colores el material de partida es el ARN. Es muy importante controlar la calidad y la cantidad del ARN, ya que puede influir directamente en la calidad final de los resultados. Para ello se utiliza un equipo llamado Bioanalyzer (Agilent Technologies, Santa Clara, California), que proporciona entre otros parámetros un valor llamado RNA Integrity Number (RIN), que sirve para decidir si la calidad del ARN es suficientemente buena como para que valga la pena usarlo para hibridar un microarray. Este valor varía entre 0 y 10 y en general suele exigirse un valor mínimo de por ejemplo 7 u 8.

Una vez decidido que la calidad del ARN de la muestra es aceptable, la cantidad inicial de ARN es amplificada unas 25 veces utilizando enzimas y cebadores específicos.

Posteriormente, el ARN es marcado con una molécula fluorescente (ficoeritrina en este caso) y fragmentado en trozos más pequeños que se puedan unir a las probes fijadas en el array mediante el proceso de hibridación. Al igual que en los arrays de dos colores, el array hibridado es iluminado en un escáner mediante un láser, para que la ficoeritrina emita luz fluorescente. La fluorescencia

registrada de cada *probe set*, es directamente proporcional a la cantidad de ARN presente en la muestra inicial de cada gen. La figura 5 es una representación esquemática del uso de microarrays de un color.

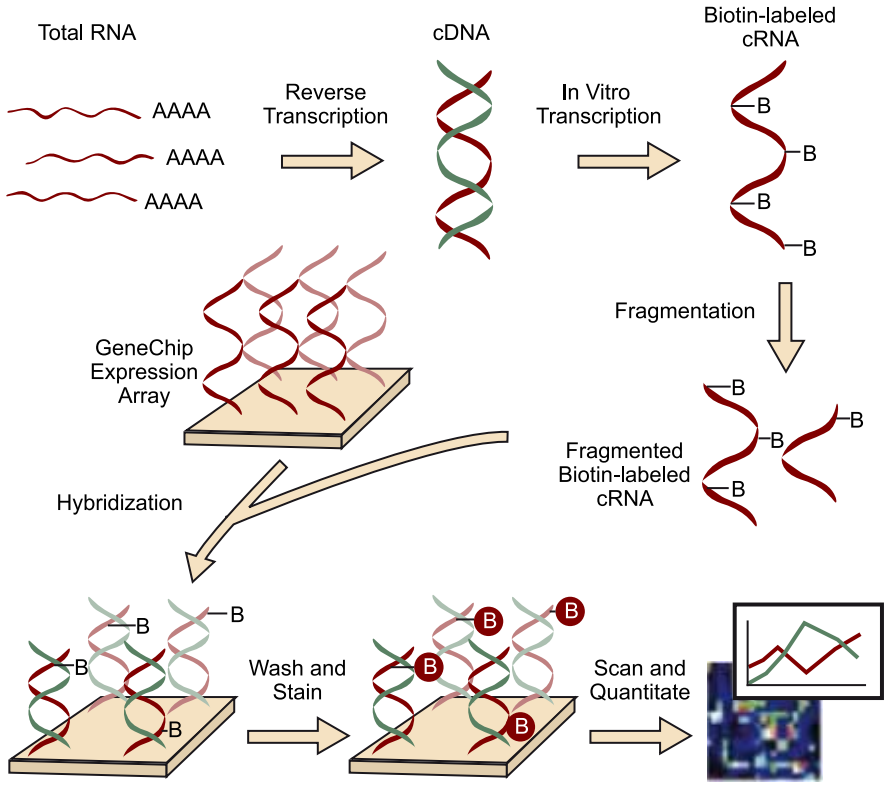

Figura 5. Esquema del funcionamiento de un microarray de Affymetrix o de un solo color

1.3.4. Cómo se mide la expresión

Los microarrays permiten cuantificar la expresión de los genes a través de la intensidad de la fluorescencia que es capturada por los escáneres. Las imágenes se convierten en valores por un proceso que no es objeto de discusión en este curso, pero que puede ser considerado relativamente fiable y estable (véase, por ejemplo [11]).

Cada tecnología genera diferentes tipos de imágenes y estas generan diferentes valores que deben ser tratados adecuadamente para proporcionar alguna clase de estimaciones de una misma variable: la expresión génica.

Tal como se ha indicado anteriormente, una de las principales diferencias entre arrays de uno y dos colores es que estos últimos se basan en la hibridación competitiva de las dos muestras mientras que los de un color solo miden cuánta muestra se hibrida con las sondas del chip. En consecuencia, en los arrays de dos colores se mide cuánto se expresa un gen en una muestra respecto a la otra, lo que tiene un sentido biológico en términos de sobreexpresión (por ejemplo, si un gen se expresa dos veces más en una condición que en la otra puede deducirse que está activado o sobreexpresado) o sub-regulación (si el gen se expresa la mitad en una condición que en otra puede deducirse que

está inhibido o regulado negativamente). En los arrays de un color en cambio tan solo se mide cuánto se expresa un gen en una escala que carece de sentido biológico.

Medición de la expresión relativa en arrays de dos colores

Cuando la imagen obtenida en un microarray de dos colores es analizada (cuantitativamente), en cada *spot* se generan algunos valores (figura 6). Aunque esto depende del software utilizado, básicamente consiste en:

- Medidas de señales, rojo (R) o verde (G) para cada canal,
- Medidas de ruido de fondo o *background*, R_b, G_b, que intentan proporcionar una medida de la fluorescencia no debida a hibridación, y
- algunas medidas de calidad para el *spot*.

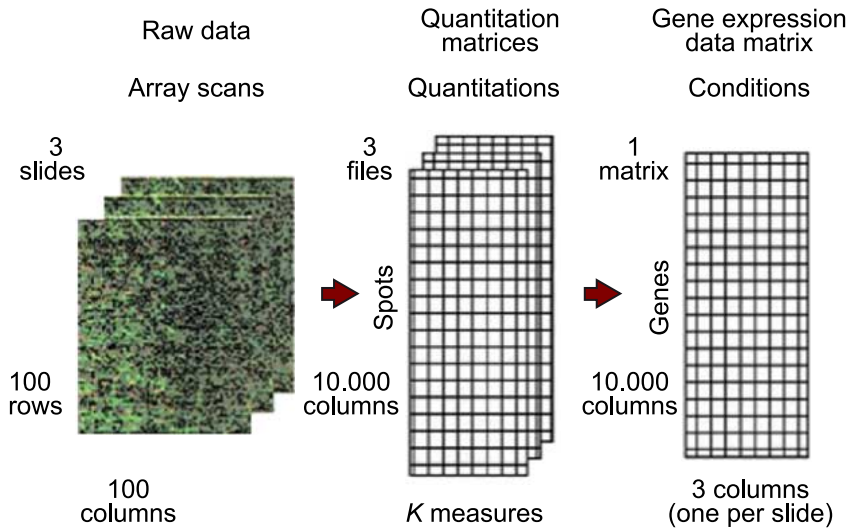

Figura 6. Esquema del tratamiento de las imágenes para su cuantificación. En el lado izquierdo se encuentran los datos de partida, tres arrays que suponemos de 100 filas y 100 columnas, es decir, 10.000 *spots*. En el centro se muestra cómo para cada *spot* se genera un cierto número de medidas que informan de diversos aspectos, que van desde la calidad del mismo a la cantidad de fluorescencia. Para cada array se genera un archivo de información con tantas filas como puntos en el array. A la derecha, finalmente, aparece la *matriz de expresión,* con una fila por cada *spot* y una columna por cada array que contiene los valores resumidos de expresión de cada gen en cada muestra.

Estas cantidades pueden utilizarse para proporcionar medidas sencillas de la razón de expresiones (en inglés *expression ratio*):

$$M = \frac{R}{G}, \qquad (1)$$

o de la razón de expresiones corregida para el ruido de fondo (en ingles *background-corrected expression ratio*):

$$M = \frac{R - R_b}{G - G_b}. \qquad (2)$$

Es habitual la utilización del logaritmo en base 2 de esta cantidad como el resultado final de expresión relativa. Esto se debe principalmente a dos motivos: por un lado, los datos de expresión se aproximan mejor con una distribución log-normal, y por otro lado, la utilización de logaritmos simetriza las diferencias haciendo más fácil la interpretación.

La tabla 1 muestra de forma simplificada el aspecto que puede tener una matriz de expresión relativa en la que para estudiar 6 muestras apareadas, tres de tejidos sanos y tres tumorales, se ha hibridado cada tumor contra su control normal dando lugar a una matriz de expresión de 5 genes y 3 columnas de expresiones relativas.

Tabla 1. Ejemplo

	log (Tum1/Norm1)	log (Tum2 /Norm2)	log (Tum3/Norm3)
Gene 1	0.46	0.80	1.51
Gene 2	-0.90	0.06	-3.20
Gene 3	0.15	0.04	0.09
Gene 4	0.60	1.06	1.35
Gene 5	-0.45	-1.03	-0.79

Ejemplo simplificado que muestra cómo podría ser una matriz de expresiones relativas obtenidas de 6 muestras apareadas (3 individuos) y 5 genes

Medición de la expresión absoluta en arrays de un color

Los arrays de Affymetrix representan cada gen como un conjunto de sondas, cada una de las cuales se corresponde con una fragmento corto de un gen. De hecho, tal como se ha dicho, no se trata de sondas sino de parejas de sondas formadas por un Perfect Match que corresponde a la cadena de ADN original y un Mismatch en las que ha cambiado su nucleótido central.

La idea subyacente en esta aproximación es que cualquiera que hibrida con la sonda de mismatch no debería representar una expresión real sino que representa lo que se denomina *background* o señal de fondo.

En los inicios de esta tecnología la compañía Affymetrix sugirió combinar ambas medidas en lo que puede verse como una medida de expresión corregida para la señal de fondo. La fórmula utilizada ha evolucionado, pero una estimación sencilla de las primeras versiones es:

$$Avg.diff = \frac{1}{|A|}\sum_{j\in A}\left(PM_j - MM_j\right), \qquad (3)$$

donde A es el conjunto de pares de sondas cuyas intensidades no se desvían más de tres veces de la desviación estándar de la principal intensidad entre todas las sondas.

La tabla 2 muestra de forma simplificada el aspecto que puede tener una matriz de expresión relativa en la que, para estudiar 6 muestras apareadas, tres de tejidos sanos y tres tumorales, se ha hibridado cada tumor y cada control normal en un array separado, dando lugar a una matriz de expresión de 5 filas (una por gen) y 6 columnas de expresiones absolutas.

Tabla 2. Ejemplo

	log (Tum 1)	log (Tum 2)	log (Tum 3)	log (Norm 1)	log (Norm 2)	log (Norm 3)
Gene 1	5.10	6.9	6.6	6.4	7.8	4.3
Gene 2	6.97	8.74	7.89	8.03	9.70	5.63
Gene 3	4.44	6.89	6.41	6.02	7.47	4.08
Gene 4	6.43	8.13	8.56	7.14	7.63	4.81
Gene 5	8.18	10.44	13.29	7.85	9.60	5.29

Ejemplo simplificado que muestra cómo podría ser una matriz de expresiones absolutas obtenidas de 6 muestras apareadas y 5 genes

La mayor diferencia entre estas dos maneras de medir la expresión no está en la fórmula específica, que ha evolucionado en ambos casos, sino en el hecho de que, mientras que en los chips de Affymetrix se tiene un único valor de expresión para cada condición, en los arrays de dos colores se trabaja con una medida de la expresión relativa entre dos condiciones. Aunque la tecnología de Affymetrix permite estimaciones más precisas, las expresiones relativas tienen una mejor interpretación intuitiva.

1.4. Bioinformática de microarrays

El aumento en el uso de los microarrays durante la primera década del siglo XXI ha ido acompañado de desarrollos metodológicos –a menudo han sido precisas nuevas técnicas para analizar los nuevos tipos de datos– así como de la creación de nuevas herramientas que implementen dichos métodos. Esto nos lleva a considerar dos aspectos muy importantes relacionados con el análisis de datos de microarrays:

1) ¿Qué programas existen para analizar los datos de microarrays?

2) ¿Qué sistemas de base de datos existen para almacenar y manipular los datos de microarrays tanto a nivel local como de forma remota?

Este tema puede ser considerado complementario, pero necesario para poner en práctica los puntos discutidos en el documento, de manera que una breve presentación de los sistemas de software y base de datos se presentan a continuación.

1.4.1. Software para el análisis de datos de microarrays

Supongamos que un estadístico o bioinformático quiere iniciarse en el análisis de datos de microarrays y después de estudiar los métodos, entiende lo que hay que hacer. Una pregunta obvia es, ¿qué herramientas puede o debe utilizar? Como la mayoría de los profesionales en el campo, está familiarizado con varios paquetes y probablemente tiene algunas preferencias.

Después de algunas búsquedas en Google, le resultará obvio que haya varias posibilidades:

- Puede utilizar los paquetes estadísticos estándar –SPSS o SAS– y analizar los datos que tendrán que haber sido procesados previamente y exportados a archivos de texto.

- Puede utilizar alguna de las muchas herramientas disponibles de forma gratuita que funcionan localmente o a través de la web.

- Puede basarse en algún proyecto desarrollado específicamente para el análisis de microarrays como *Bioconductor*.

- Puede comprar alguno de los programas comerciales existentes.

Como es habitual, cada opción tiene aspectos positivos y negativos. El uso de paquetes estadísticos estándar –SPSS o SAS– tiene la curva de aprendizaje más corta, pero no permite hacer la mayoría de los preprocesamientos habituales, tales como la normalización o resumen, por lo que debe combinarse con otro software. Además, si lo que uno quiere hacer es aplicar métodos específicos, que se estudiaran más adelante en el texto, como por ejemplo *limma* o *FDR*, los paquetes estadísticos estándar resultan claramente insuficientes. Algunos paquetes de estadística como `S+` o SAS han desarrollado extensiones para el análisis de datos de microarrays.

Software libre o de código abierto

Existen numerosas herramientas gratuitas para el análisis de microarrays. Algunas de ellas, además, ofrecen la posibilidad (libertad) de modificar el programa porque incluyen el código fuente de programa (por tanto, se las llamas de software de código abierto o de software libre). Y otras, en cambio, ponen restricciones a cómo permiten usar el programa, restringen la libertad de modificar el código fuente del programa, y por tanto, aunque sean gratuitas, se las llama de software propietario, no-libre o cerrado, por contraposición al software anterior.

Cualquiera de estas herramientas gratuitas puede conllevar un esfuerzo considerable en aprender a utilizarlas, pues tienen diferentes grados de madurez y de facilidad para que nuevos usuarios las utilicen. Asimismo, puede pasar que

estas herramientas no sigan estándares comunes de organización de los datos o flujos de trabajo, lo que hace que el aprendizaje de uno no suele ayudar al aprendizaje de otro. Y cabe la posibilidad de que, como herramientas gratuitas o demasiado jóvenes, presenten una mayor tasa de errores que la deseada. En cualquier caso, a menudo pueden resultar útiles para un análisis exploratorio de nuestros datos de microarrays, o para el mundo de la enseñanza, en especial si hay una comunidad de usuarios lo suficientemente grande de esa herramienta para garantizar que no hay demasiados problemas básicos con su uso. Pero si se desea usarlos repetidamente para la realización de estudios de media a alta complejidad, pueden resultar ser insuficientes, ya sea porque carecen de métodos, porque son ineficientes o simplemente porque no tienen la capacidad de programación para automatizar tareas repetitivas.

A pesar de estas críticas, los programas libres especialmente (más allá de los únicamente gratuitos), pueden ser una forma suave para introducirse en el tema de los análisis de datos de microarrays. A continuación comentamos brevemente algunas de nuestras herramientas libres favoritas:

- **BRB array tools**. Es un complemento de Excel (tambien llamado *add-inn*) que combina **R**, C y Java para hacer los cálculos y utiliza Excel para interactuar con el usuario –lo que significa que solo está disponible para usuarios de Windows. El programa ha sido desarrollado y es mantenido por un equipo del *Biometrics Research Branch* del Instituto Nacional del Cancer (*NCI*) de Estados Unidos. Resulta muy atractivo a primera vista, con un gran número de opciones, tanto para análisis comparativos como para estudios de clasificación. Los resultados que proporciona también son claros e intuitivos Además, en su página web existe una base de datos con ejemplos reales listos para ser descargados y analizados con el programa. Como puntos débiles cabe destacar que originalmente estaba pensado para arrays de dos colores, lo que se hace patente en la estructura del programa y que la creación de un nuevo estudio desde cero, importando los datos originales, no es todo lo flexible que uno desearía. Lo peor del programa es que cuando falla, debido a algún problema en su código de origen, resulta bastante difícil conseguir que cese el error, especialmente si se utiliza en ordenadores con versiones españolas de Windows.

- **TM4**. Es un conjunto de cuatro programas de código libre escritos en Java desarrollados por el instituto TIGR (ahora J. Craig Venter). El programa de análisis, denominado *MeV* por *MultiExpressionViewer* es muy bueno y bastante robusto (se bloquea mucho menos que BRB) y ofrece capacidades de análisisde múltiples tipos de datos de expresión –microarrays o RNA-seq– u otros datos comunes en estudios de genómica como aCGH o ChIP-seq. Los módulos restantes se complementan bien con el de análisis –que es el que más ha evolucionado–, existiendo un programa para análisis de imagen (Spotfinder), uno para la normalización de arrays de dos colores (MIDAS), otro en forma de aplicación web para la normalización de arrays de Affymetrix (AMP) e incluso un sistema de bases de datos (MADAM) para

almacenar los datos de los estudios, desde las muestras a los resultados de la hibridación y los análisis.

- ***Expression console*** de Affymetrix. A menudo el punto más delicado del análisis de microarrays es el preprocesado, es decir, el paso de las imágenes a la matriz de expresión. Algunos programas de análisis permiten llevar a cabo el preprocesado mientras que otros lo implementan de forma parcial o no lo incluyen en absoluto. El programa *Ezpression Console* de Affymetrix es una herramienta sencilla, descargable de forma gratuita (aunque es preciso registrarse) de la web de Affymetrix, que funciona en Windows y permite realizar un control de calidad básico, así como la normalización y sumarización (descrita en capítulos posteriores) de cualquier array producido por Affymetrix.

- ***Suite*** **de análisis de expresión o Babelomics**. Es un conjunto integrado de herramientas para el análisis de datos de microarrays (y otros tipos de datos que ha ido incorporando con el tiempo) disponible en la web. Babelomics ha sido diseñado para proporcionar una interfaz intuitiva basada en web que ofrece diversas opciones de análisis desde la etapa inicial de preprocesamiento (normalización de Affymetrix y experimentos de microarrays de dos colores y otras opciones de preprocesamiento), hasta el paso final de la elaboración de perfiles funcionales que sirven de soporte para la interpretación biológica de los resultados. El programa –la aplicación web, de hecho– es muy completo, está continuamente mejorando y actualizándose y si tan solo se le tuviera que encontrar un inconveniente, sería que solo está disponible como aplicación web, lo que puede hacer que ciertos procesos de carga o descarga o algunas colas de ejecución resulten lentos en ocasiones. A pesar de ello, es la herramienta más completa de todas las que hemos comentado.

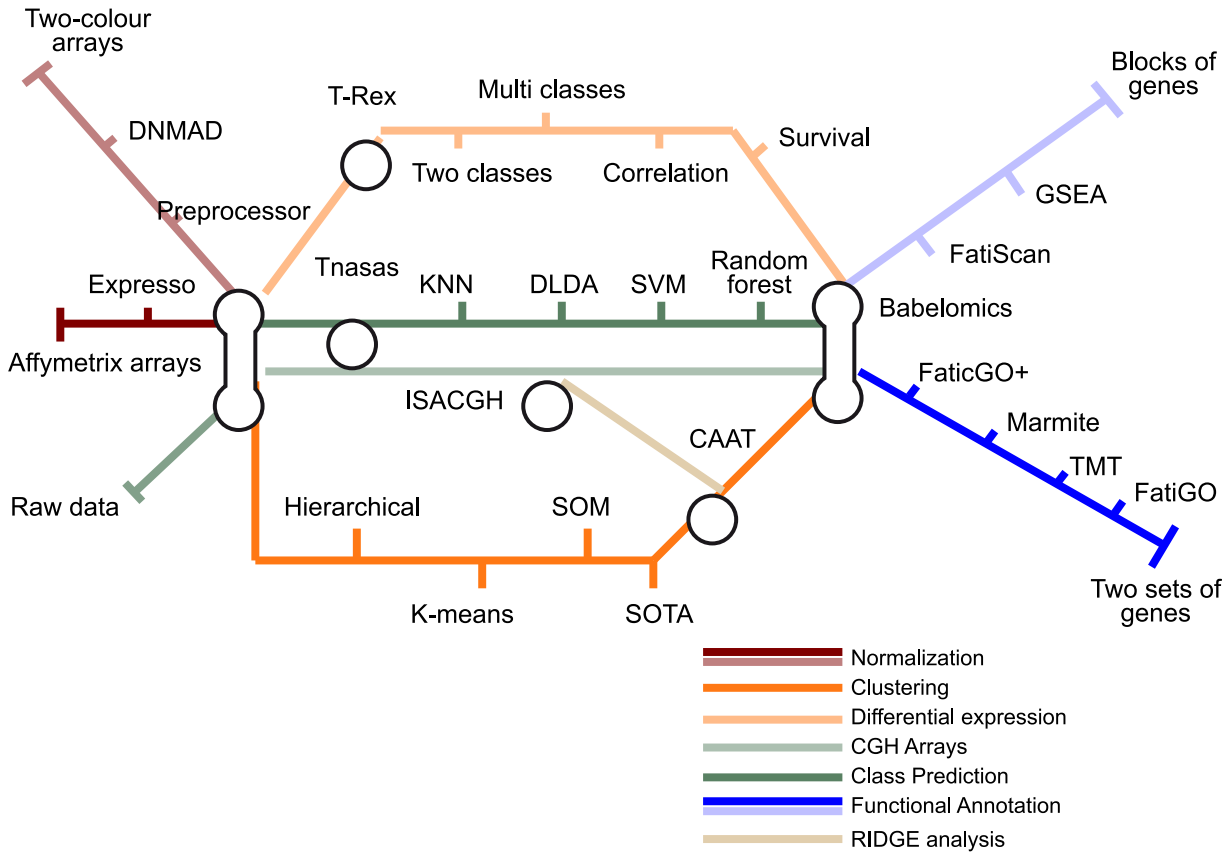

Figura 7. Software de análisis de la expresión génica o Babelomics
Un mapa de las funcionalidades de Babelomics organizados como en una línea de metro. Un usuario normalmente debe comenzar en alguna parte de la izquierda del mapa y finalizar en algún lugar de la derecha.

El proyecto Bioconductor

Una de las opciones para el análisis de los datos mencionados anteriormente es la combinación de un software estándar, tales como `Matlab`, `Mathematica` o **R** con librerías específicas diseñadas para el análisis de microarrays. Aunque existen algunas extensiones de Matlab para el análisis de microarrays, es con **R** con la que esta complementariedad ha alcanzado dimensiones inesperadas. El proyecto Bioconductor comenzó en el 2001 como un proyecto de código abierto y libre de desarrollo de software para el análisis de datos genómicos. Su gran éxito le ha hecho crecer a partir de poco más de una docena de paquetes a cientos de ellos. Casi todas las técnicas disponibles en el análisis de microarrays tienen su propio paquete, y con frecuencia hay varios de ellos.

La gran potencia de este proyecto implica también algunos de sus inconvenientes: en primer lugar, al ser un proyecto de código abierto significa que los desarrolladores aportan sus programas tal como cada uno cree o sabe que se tiene que construir. Aunque hay sistemas de control para evitar la no ejecución del código, es más difícil de garantizar (a excepción de la honestidad de los desarrolladores) que se ejecuta como se indica. El poder del Bioconductor también se basa en la flexibilidad del lenguaje **R**. Esto hace que los usuarios que no tienen un nivel medio o alto de utilización de **R** no puedan aprovechar todas las posibilidades del proyecto.

A pesar de estas aparentes dificultades, Bioconductor es la herramienta elegida por muchos estadísticos y/o bioinformáticos, y la razón principal es que, cuando uno ha sido capaz de sentirse cómodo con ella, su potencia y flexibilidad es difícil de igualar. El hecho de que esté implementado en **R** hace que sea posible implementar cualquier análisis que se desee realizar en forma de *scripts*, es decir, que potencia la reutilización del código y permite la automatización de tareas repetitivas.

El software propietario

Hay muchas herramientas comerciales disponibles para el análisis de microarrays de datos. Estos van desde pequeños programas específicos de un tipo de datos hasta paquetes de software grandes, como Partek Genomics Suite que es una solución completa optimizada para los cálculos eficientes y rápidos, así como para la mayoría de los datos genómicos existentes. Los programas comerciales –como contrapuestos a los de código abierto y los gratuitos (que no debemos olvidar que no es lo mismo)– presentan algunas ventajas y algunos inconvenientes razonables. Por un lado pueden ser caros y con restricciones (por ejemplo, ciertas licencias permiten que tan solo los utilice un usuario cada vez). Por el otro, pueden resultar más homogéneos o robustos que ciertos proyectos o tener un sistema de soporte que haga que los potenciales usuarios los elijan como opción a pesar de su precio.

1.4.2. Bases de datos de microarrays

La diversidad de formatos y tipos de experimentos de microarrays ha dificultado que se haya impuesto un formato de base de datos, por lo que, a pesar de haberse realizado miles de estudios y de ser sus datos públicos, no existe ningún sistema estándar para almacenar o publicar datos de microarrays –ni obviamente de otros tipos de datos genómicos.

Existe, eso sí, un estándar sobre qué información debe proporcionarse respecto a un estudio llevado a cabo con microarrays, denominado estándar MIAME (por *minimum information about a microarray experiment*) pero en realidad se trata de algo tan básico que apenas resulta de utilidad.

Cuando hablamos de sistemas de almacenamiento de datos de microarrays, podemos hacerlo a dos niveles distintos: sistemas de bases de datos locales, normalmente para almacenar nuestros propios datos, o grandes repositorios donde depositar y acceder a los datos de estudios publicados. Se pueden distinguir dos niveles en los que los sistemas de bases de datos se han desarrollado.

1) Sistemas de bases de datos locales. El análisis de los datos de microarrays pasa por una serie de pasos y en cada uno de ellos se necesitan o se generan distintos tipos de datos, que pueden ser imágenes, archivos binarios o archivos de texto. Pocos sistemas permiten almacenar todos estos datos de forma flexible y fácilmente accesible. Entre los que han alcanzado cierta populari-

dad, se encuentra BASE (base.thep.lu.se) o caArray (http://cabig.cancer.gov/solutions/applications/caarray/). A pesar de su evidente interés, son sistemas difíciles de hacer funcionar y de adaptar a las necesidades de cada caso concreto, por lo que muchos usuarios y centros suelen preferir algún sistema hecho a medida y probablemente menos flexible que los citados.

2) Repositorios públicos de arrays. La comunidad biológica se ha comprometido, desde el inicio de los microarrays, a que los datos de los experimentos publicados deben hacerse públicos. Esto ha creado la necesidad de repositorios de microarrays públicos, donde cualquier usuario puede de`psotar sus datos en una forma adecuada. Al mismo tiempo, se ha creado una impresionante cantidad de datos disponibles para un nuevo análisis para cualquiera que desee hacerlo, que ofrece una riqueza sin precedentes de oportunidades cuyo poder está empezando a mostrar. Entre los principales repositorios de datos de microarrays podemos destacar el Gene Expression Omnibus, desarrollado y mantenido por el Instituto nacional de la Salud de Estados Unidos y ArrayExpress desarrollado y mantenido por el European Bioinformatics Institute. Ambos sistemas contienen los datos de miles de experimentos y son relativamente sencillos de usar, tanto para depositar datos como para acceder a ellos.

1.5. Extensiones (1): Otros tipos de microarrays

Este curso se centrará en torno al tipo más popular de microarrays: los microarrays de expresión de ARN, diseñados para estudiar la expresión génica basada en la información sobre la cantidad de ADN que se transcribe a ARNm.

El desarrollo de la tecnología de los microarrays ha conllevado que, en paralelo a los arrays de expresión génica, se hayan podido desarrollar otros tipos de microarrays. Por "otros" pueden entenderse distintas cosas. Algunos se basan también en ADN o ARN y permiten estudiar otro tipo de problemas como el número de copias o los polimorfismos de un solo gen.

Otros se han desarrollado pensando en otras sustancias, como los arrays de proteínas o de carbohidratos. Una descripción completa de cada tipo, su uso, objetivos y análisis de los datos está fuera del alcance de este curso. Sin embargo, para dar un ejemplo de las similitudes y diferencias entre los microarrays de expresión y las tecnologías relacionadas hacemos una breve reseña de los problemas que requieren de estas tecnologías alternativas y haremos una breve descripción de uno de ellos, los arrays de genotipado o de SNPs.

1.5.1. Otros microarrays de ADN

Uno de los objetivos principales de la genómica funcional es la comprensión y la curación de la enfermedad. Se sabe que en muchas alteraciones genéticas subyacen anomalías y/o enfermedades. Por ejemplo:

- Mutaciones puntuales –cambio de una o más bases– puede dar lugar a proteínas alteradas o a cambios en el nivel de expresión.

- La pérdida de copias de genes puede reducir el nivel de expresión. Estos cambios pueden estar relacionados, por ejemplo, con la supresión tumoral.

- El aumento del número de copias de un gen puede hacer que aumente el nivel de expresión génica, lo que se puede relacionar con la activación de ciertos oncogenes.

- La metilación o desmetilación de la región promotora de los genes puede relacionarse con la disminución o el aumento del nivel de expresión de ciertos genes y por tanto, tener un papel importante en procesos como la activación o supresión tumoral.

Existen distintos tipos de microarrays para estudiar las manifestaciones y los efectos de estas alteraciones. Los puntos planteados en el párrafo anterior podrían ser estudiados con arrays de genotipado o de SNP (se pronuncia *esnips*), con arrays de hibridación genómica comparativa (aCGH) o con arrays de metilación.

Arrays de genotipado

El polimorfismo de un solo nucleótido (en ingles *single nucleotide polymorphism* o *SNP*) es una forma de mutación puntual que consiste en un cambio en alguno de los cientos o miles de millones de base que forman el genoma. Estas mutaciones puntuales se encuentran dispersas al azar por todo el genoma. Miles de polimorfismos de un nucleótido han sido –y siguen siendo– identificados como parte de los proyectos de secuenciación. En contra de lo que uno podría pensar –por su aparentemente pequeño tamaño– muchas de estas mutaciones se han conservado durante la evolución y dentro de las poblaciones, por lo que pueden ser utilizados como marcadores genéticos o más exactamente genotípicos.

Los arrays de SNP fueron creados para permitir detectar polimorfismos dentro de las poblaciones. Se basan en los mismos principios que los arrays de expresión, pero cada sonda está diseñada para detectar las diferentes variantes de polimorfismo de un solo nucleótido (es decir A, C, T o G) para cada SNP conocido.

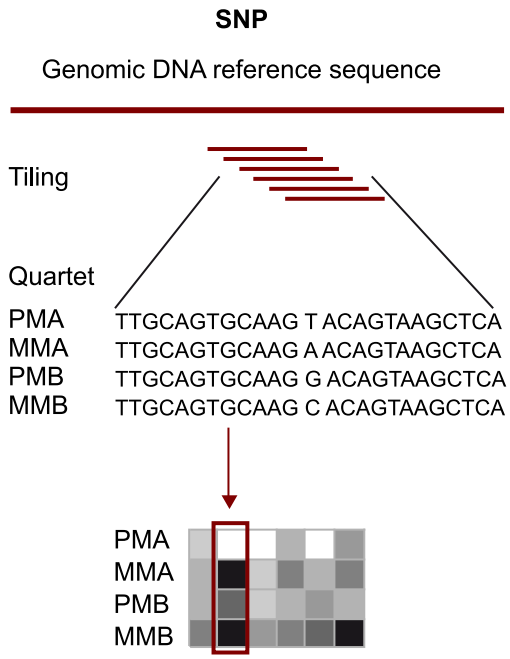

Figura 8. Utilización de arrays de genotipado
Explicación simplificada de la utilización de arrays de SNP para detectar polimorfismos de nucleótido único.

Los arrays de SNP tienen muchas aplicaciones. Entre ellas se puede destacar:

- **Estudios basados en el linaje familiar**. El ADN de los familiares afectados con una enfermedad en particular puede ser comparado con el ADN de los miembros de la misma familia que no tienen esta condición. Estos estudios permiten identificar las diferencias genéticas que pueden estar asociadas con la enfermedad.

- **Estudios de asociación a nivel poblacional**. Consiste en determinar las diferencias en las frecuencias de SNP en los individuos afectados y no afectados en una población. El objetivo es identificar SNP en particular o combinaciones de SNP que difieren entre los dos grupos, y por lo tanto, se pueden considerar asociados con la enfermedad. Estos estudios requieren un gran número de muestras para representar adecuadamente a la población. Esta es una de las aplicaciones más conocidas de este tipo de arrays que ilustra cómo pueden ser utilizados en la identificación de genes relacionados con enfermedades complejas.

- **Cambios del número de copias**. Los arrays de SNPs pueden ser también utilizados para estudiar las regiones con variabilidad del número de copias –una variante del número de copias (CNV) es un segmento de ADN que es de 1 KB o más grande y está presente en un número variable de copias en comparación con un genoma de referencia. La identificación de los cambios de número de copias es útil para detectar tanto las aberraciones cromosómicas como la pérdida del número de copias neutrales de heterocigosidad (LOH), eventos que son característicos de muchos tipos de cáncer.

1.5.2. Otros tipos de microarrays: proteínas o carbohidratos

Existe un amplio consenso sobre el hecho de que la información obtenida de los microarrays de ADN no es suficiente para alcanzar una completa comprensión de los procesos celulares, la mayoría de los cuales son controlados por las proteínas que interactúan con otras moléculas, como los hidratos de carbono, a menudo implicados en importantes mecanismos biológicos como la interacción de patógenos con el huésped, el desarrollo o la inflamación. Los microarrays de proteínas y carbohidratos son dos ejemplos de la extensión del uso de estas herramientas para el análisis de alto rendimiento de diferentes tipos de moléculas. Los microarrays de tejidos son un tipo diferente de extensión en la que el soporte experimental no está formado por distintas variantes de un solo tipo de molécula, sino de un tipo de tejidos.

1.6. Extensiones (2): *NGS: Next(Now) Generation Sequencing*

A finales de la primera década del siglo XXI parece ser que la revolución de los microarrays está llegando a su fin ([8], [5]) y una nueva tecnología está inundando las revistas y congresos científicos. Se trata de la ultrasecuenciación también llamada *next generation sequencing* o también, jugando con las siglas y con la idea de que la técnica ya está en uso, *now generation sequencing*.

Básicamente, la ultrasecuenciación permite hacer lo mismo que la secuenciación tradicional o Sanger, es decir, obtener la secuencia de una cadena de ADN o ARN que se ha preparado previamente para ello mediante creación de un cierto número de copias o "librerías". La diferencia, una vez más, está en la cantidad de secuencias que es posible obtener. Mientras que la secuenciación tradicional suele poder producir entre varios cientos y varios miles de pares de bases por día, la capacidad de los nuevos métodos es de órdenes de magnitud superior, lo que significa el acceso rápido y económico a la capacidad de secuenciar genomas de eucariotas en un tiempo breve –semanas o días en el 2012– a un coste ínfimo de secuenciar. De hecho, a principios del año 2012 es posible secuenciar un genoma humano en menos de un mes por menos de 10.000 euros, pero ya hay compañías que anuncian el genoma de 100 euros en 24 horas para antes de un año.

Este gran incremento en la capacidad de producir secuencias ha representado, de nuevo, un cambio de paradigma en la resolución de muchos problemas científicos. Hoy es posible considerarla como una aproximación directa al estudio de múltiples problemas y la secuenciación se aplica a un gran número de campos: desde la metagenómica, que estudia a nivel genómico la diversidad de los ecosistemas bacterianos como lagos o intestinos, al análisis de variantes estructurales en exomas o genomas y su posible asociación con enfermedades hereditarias como el autismo o el cáncer de mama. Una variante de la secuenciación de ADN ha sido el RNA-seq, que secuenciando ADN complementario permite abordar un estudio mucho más fino de la expresión génica de la que se pueda hacer con microarrays, dado que permite cuantificar directamente el

producto de la expresión -podemos secuenciar el ARN y contar los tránscritos para saber cuánta expresión se ha dado- a la vez que es posible obtener información acerca de secuencias no identificadas previamente -por lo que no se habían podido poner en los microarrays.

En este subapartado se presenta una visión general de la ultrasecuenciación, las tecnologías que funcionan en el 2011, los problemas bioinformáticos y computacionales que aparecen y cómo se resuelven y algunos de los problemas que pueden abordarse con estas técnicas, con un sesgo hacia los aspectos bioinformáticos y de análisis de datos que no dejan de ser el objeto de este curso.

1.6.1. Visión global de las tecnologías de secuenciación

El ADN no puede ser secuenciado de golpe, ha de ser fragmentado suficientemente, secuenciado a trozos y finalmente reensamblado. La característica básica presente, tanto en el enfoque tradicional (Sanger) como en ultrasecuenciación, es que, una vez fragmentado el ADN a secuenciar, cada fragmento deberá ser amplificado o clonado. Debido a este paso previo de fragmentación y amplificación, se se suele denominar a las tecnologías de secuenciación en general con el término *shotgun sequencing* o *shotgun cloning*.

1.6.2. De la secuenciación Sanger a la ultrasecuenciación

Desde principios de los años noventa la secuenciación de ADN se ha realizado casi exclusivamente mediante secuenciación Sanger más o menos automatizada, pero con muy poca capacidad de procesamiento en paralelo [4, 5], es decir, no era posible secuenciar varios fragmentos a la vez sino que se tenían que obtener secuencialmente. La clave principal del método de Sanger artesanal fue el uso de didesoxinucleótidos trifosfato (ddNTPs) como terminadores de cadena, motivo por el cual también se hace referencia a este con el término *chain termination method*. Para la lectura de un solo fragmento, deben realizarse por separado cuatro mezclas de reacción. Cada mezcla de reacción contiene los cuatro nucleótidos trifosfato (dATP, dCTP, dTTP, dGTP), ADN polimerasa I, un cebador o *primer* marcado radioctivamente (permite el acoplamiento de la polimerasa e inicio de la reacción en ese punto) y un didesoxinucleótido ddNTP (A, C, G, o T), a una concentración baja. El didesoxinucleótido utilizado competirá con su homólogo por incorporarse a los clones del fragmento dado, produciendo la terminación de la síntesis en el momento y lugar donde se incorpora, ya que al carecer de grupo 3'-OH no es posible el enlace fosfodiester con el siguiente nucleótido. Por este sistema, en cada mezcla de reacción se producen una serie de moléculas de ADN de diferente longitud que terminan todas en el mismo tipo de nucleótido y marcadas todas radiactivamente por el extremo 5' (todas contienen en el extremo 5' el *primer* utilizado).

Las secuencias de ADN sintetizadas por este procedimiento y obtenidas en cada mezcla de reacción se separan por tamaños mediante electroforesis en geles verticales de acrilamida muy finos (0.5 mm de espesor) y de gran longitud (cerca de 50 cm) que permiten distinguir secuencias de ADN que se diferencian en un solo nucleótido. Los productos de cada una de las cuatro mezclas de reacción se insertan en cuatro carriles diferentes del gel.

Una vez terminada la electroforesis, el gel se pone en contacto con una película fotográfica de autorradiografía. La aparición de una banda en una posición concreta de la autorradiografía en uno de los cuatro carriles nos indica que en ese punto de la secuencia del ADN de nueva síntesis (complementario al ADN molde) está la base correspondiente al didesoxinucleótido utilizado en la mezcla de reacción correspondiente.

Teniendo en cuenta que el ADN de nueva síntesis crece en la dirección 5' -3', si comenzamos a leer el gen por las secuencias de menos tamaño (extremo 5') y avanzamos aumentando el tamaño de las secuencias (hacia 3'), obtendremos la secuencia completa del ADN de nueva síntesis en la dirección 5' -3' del fragmento dado.

Después de tres décadas de perfeccionamiento gradual, la secuenciación Sanger permite alcanzar longitudes de lectura de hasta 1000 pares de bases ("bp"). La ventaja principal de este método es su elevada fiabilidad –se suele considerar que posee una precisión del 99,999%, es decir, una tasa de errores de 0,001%. La contrapartida es que el coste es relativamente elevado, de unos 0,5$ por 1.000 bases, lo que en una grosera aproximación representaría unos 1,5 millones de $ para secuenciar un genoma humano.

1.6.3. Tecnologías de ultrasecuenciación

Existen distintas formas de llevar a cabo la ultrasecuenciación. En el párrafo siguiente se discute brevemente lo que se conoce como *cyclic array-sequencing* y, que comprende las principales tecnologías que se están aplicando en la actualidad, es decir: la secuenciación 454 (usada en los *454 genome sequencers*, de Roche Applied Sciencel), la tecnología Solexa (usada en los equipos Hi-Seq de la compañía Illumina), la plataforma SOLiD (de la compañía Applied Biosystems) y otras de más reciente aparición, como la tecnología *Polonator* de la compañía Dover/Harvard y la *single molecule sequencer* de la empresa Helicos.

Aunque las plataformas mencionadas difieren en su bioquímica, el flujo de trabajo es conceptualmente similar (figura 9*a*). Hay varias aproximaciones para la generación de los grupos de copias (*polonies*), pero todas ellas suponen que al final los derivados PCR de una única molécula de la librería acaben agrupados y reunidos en la misma localización espacial. Entre ellas están las *in situ polonies* (sujeción de localización única), *bridge* PCR (sujeción a sustrato plano) y *emulsion* PCR (sujeción a gotas de tamaño entorno a una micra, de-

nominadas *beads*). El proceso de secuenciación (figura 9*b*) consiste en general en alternar ciclos de irrigación enzimática con adquisición de datos basada en procesado de imagen.

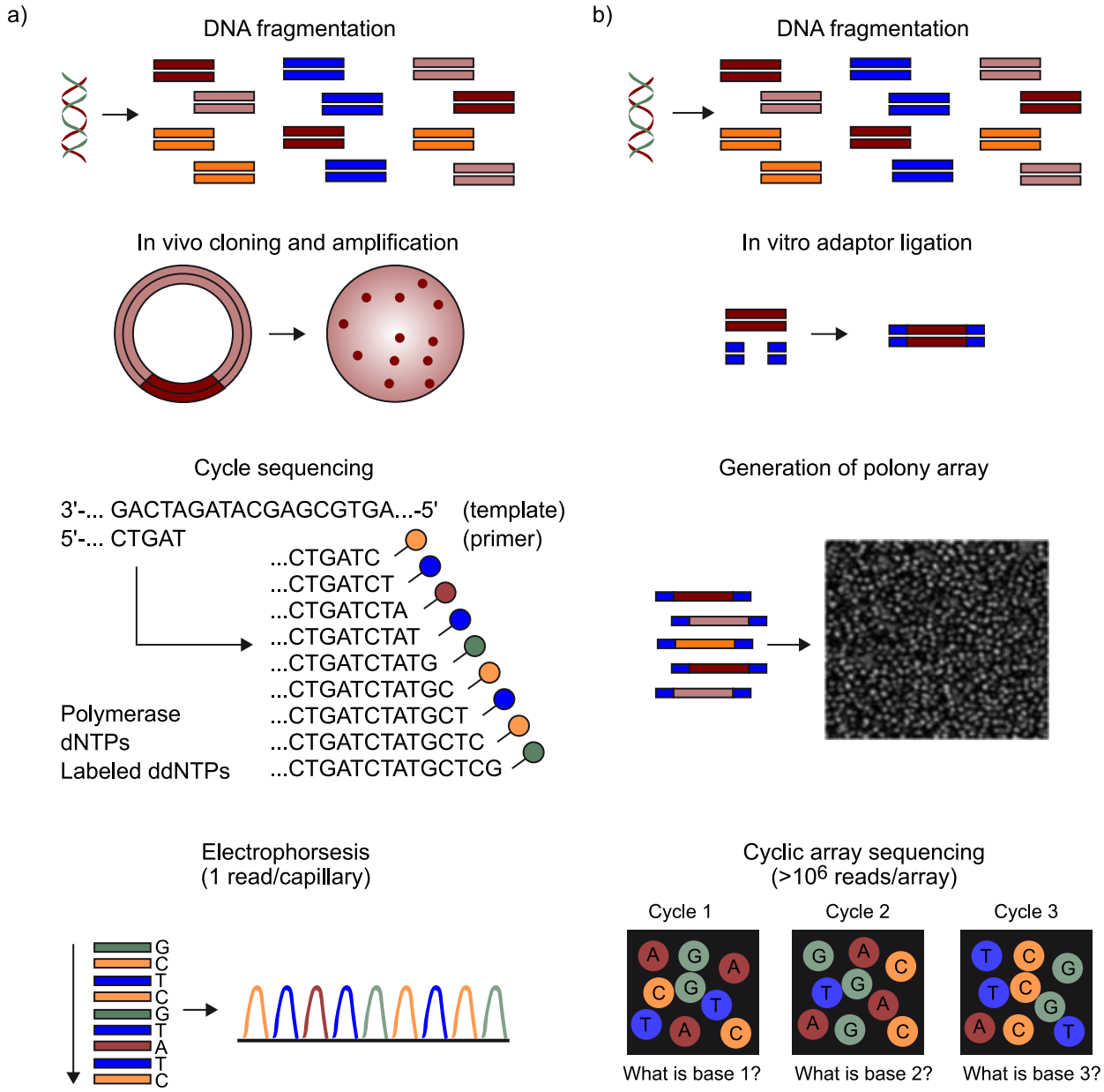

Figura 9. Comparación entre las secuenciación Sanger (izquierda) y la ultrasecuenciación (derecha).

2. El lenguaje estadístico R

2.1. ¿Qué son R y Bioconductor?

R es un entorno de programación diseñado específicamente para trabajos estadísticos. Se trata de un proyecto de software libre, resultado de la implementación GNU del lenguaje S. R es, probablemente, uno de los lenguajes más utilizados por la comunidad estadística, siendo además muy popular en el campo de la investigación biomédica, la bioinformática y las matemáticas financieras. Se trata, además, de un programa modular, lo que significa que los usuarios pueden escribir extensiones y distribuir los códigos a otros usuarios. El código se distribuye en forma de librerías listas para su utilización que se denominan paquetes. La mayor parte de la funcionalidad de R está en los paquetes, no solo en los realizados por los desarrolladores del programa madre sino también los realizados por otros usuarios. Por su parte, Bioconductor es un proyecto de código abierto para el análisis de datos de genómica, que contiene un repositorio de paquetes de R para bioinformática. Sus orígenes se remontan al otoño del 2001 con el objetivo de desarrollar e integrar software para el análisis estadístico de datos de laboratorio en biología molecular. Su principal aplicación es el análisis de microarrays.

2.1.1. Descarga e instalación de R

R se distribuye bajo la licencia GNU GPL y está disponible para los sistemas operativos Windows, Macintosh, Unix y GNU/Linux. Además, también están disponibles los códigos fuente para la compilación de otros sistemas operativos. La página de descarga del programa R se encuentra en http://www.r-project.org. Se busca el programa de instalación del sistema que nos interesa y se descarga al ordenador. Se instala el programa base de R y se recomienda también la instalación de algunos paquetes. En los siguientes subapartados se describe la instalación detallada en los diferentes sistemas operativos.

Instrucciones para la instalación en Windows

Para instalar R en Windows basta con instalar el ejecutable que aparece en la web: http://cran.es.r-project.org/bin/windows/base/. Para ello, descargad el archivo al ordenador para posteriormente ejecutarlo. El programa de instalación se ejecuta mejor con privilegios de administrador. Aceptad la licencia de uso y seleccionad la carpeta de instalación. Tened en cuenta que R requiere 1 GB de memoria disponible en el equipo para su correcto funcionamiento.

Instrucciones para la instalación en Mac OSX

El directorio `bin / macosx` del repositorio principal de R llamado CRAN contiene un paquete estándar de Apple para instalar R llamado `R.dmg`. Una vez descargado y ejecutado, el programa instalará la versión actual de R para no desarrolladores. `RAqua` es una versión nativa de R para Mac OS X de Darwin con un R.app Mac OS X GUI. En el interior de `bin/macosx/powerpc/contrib/x.y` hay paquetes precompilados binarios (para la versión PowerPC de Mac OS X) para ser utilizado con la RAqua correspondiente a la versión de R "x.y". La instalación de estos paquetes está disponible a través del menú Package de la interfaz gráfica de usuario R.app.

Instrucciones para la instalación en UNIX y GNU/Linux

El código siguiente es una muestra de cómo realizar una instalación básica en un sistema GNU/Linux desde los repositorios habituales en la distribución Debian (Ubuntu y similares):

```
sudo apt-get install r-base r-base-dev r-recommended
```

El equivalente para distribuciones de GNU/Linux basadas en Fedora/RHEL sería:

```
yum install R R-devel R-lattice
```

Instrucciones para la instalación de Bioconductor

Para instalar Bioconductor debemos tener instalado de forma correcta el programa R. La forma más sencilla de proceder es usando un *script* de instalación ya creado. En primer lugar, se debe descargar y ejecutar el *script* desde una sesión de R:

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

Si no se puede acceder a Internet directamente desde R, pueden descargarse los paquetes desde el sitio web de Bioconductor. Si no sabe cómo descargar paquetes adicionales en el programa R, puede consultar el siguiente subapartado.

Instalación de paquetes adicionales en cualquier sistema

Todos los paquetes de R y Bioconductor se almacenan en Internet en lugares (servidores) accesibles libremente que se denominan repositorios. `CRAN` es el repositorio central para los paquetes de R en general, pero los paquetes Bioconductor se almacenan en un repositorio web propio. Los paquetes de los repositorios de CRAN o Bioconductor se pueden instalar directamente desde R.

En primer lugar se debe seleccionar el repositorio desde R:

```
> setRepositories()
```

Esto nos da una lista de repositorios disponibles. Debemos seleccionar al menos uno y especificar:

```
> install.packages()
```

Bioconductor también contiene muchos paquetes de anotación, como KEGG y como GO. Los paquetes disponibles se pueden comprobar en la web de Bioconductor. Se puede seleccionar la última versión de la lista de actualizaciones. Los nombres de los paquetes se deben especificar tal como aparecen en la lista de la página web. Un ejemplo sería el siguiente:

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite(c("KEGG", "GO","cMAP"))
```

Se pueden instalar al mismo tiempo un número ilimitado de paquetes con esta sintaxis. Solo se tienen que añadir los nombres de los nuevos paquetes entrecomillados y separados por comas (,).

Entre los paquetes de anotación destacan por ejemplo las anotaciones para todos los chips de la compañía Affymetrix.

Por ejemplo, para instalar toda la información relacionada con la levadura ygs98 se debe escribir:

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite(c("ygs98", "ygs98cdf", "ygs98probe"))
```

Estos paquetes contienen las anotaciones de las sondas o probes (`ygs98`), las asignaciones entre las sondas y grupos de sondas (probesets) (`ygs98cdf`), y las secuencias de la sonda (`ygs98probe`).

Instalación de paquetes adicionales de archivos zip o tar.gz

Si R no permite la conexión a Internet, se pueden descargar los paquetes en el equipo local para ser instalados posteriormente. Los paquetes para Windows son archivos .zip y los paquetes para sistemas UNIX o Mac OSX son archivos con extensión .tar.gz. En Windows, un paquete zip se puede instalar desde el menú de interfaz de usuario Packages usando la opción `Install package(s) from local zip files`. Para los sistemas UNIX, la instalación es un poco más complicada. Se utiliza el comando `install.packages()`. Por ejemplo, para instalar un paquete llamado mi_archivo el comando sería:

```
> install.packages(repos = NULL, Pkgs = "mi_archivo.tar.gz")
```

Antes de ejecutar el *script* de instalación, es necesario cambiar la ruta de R para seleccionar la carpeta donde se encuentra el paquete. Esto se hace con el comando setwd (), y la ruta se introduce entre comillas, por ejemplo, `setwd("/uoc/curso")`.

2.1.2. Interfaces gráficas de usuario

Existen varias interfaces gráficas de usuario en R, que van desde editores de texto integrados con R hasta interfaces gráficas del estilo apuntar y clicar. Uno de ellos es RStudio.

RStudio es un entorno de desarrollo integrado (IDE) de R con una interfaz de usuario muy intuitiva y que tiene potentes herramientas de codificación. Está disponible para todas las plataformas incluyendo Windows, Mac OS X y GNU/Linux. Al igual que R, está disponible bajo una licencia de software libre que garantiza la libertad de compartir y modificar el software.

2.2. Introducción al lenguaje R

2.2.1. Conceptos básicos del lenguaje R y su entorno

Empezar con R sin ningún conocimiento previo es una tarea bastante exigente. En este subapartado se ofrece una visión general de los fundamentos de R y de algunos de los comandos R más comunes.

Iniciar y cerrar R

R se puede iniciar haciendo clic en su icono (Windows y Mac OS X) o escribiendo su nombre en el símbolo del sistema (GNU/Linux y UNIX). R se cierra con el comando `q()`. Cuando se cierra R, el usuario tiene la opción de guardar todo lo que ha hecho en la misma sesión. R pregunta si se desea guardar el espacio de trabajo (`Save workspace image`). Si la respuesta es sí, todos los objetos se guardan en un archivo llamado `.RData`, y el historial de comandos se guarda en un archivo llamado Rhistory. Esto es muy útil, ya que estos archivos pueden cargarse de nuevo en la memoria y el análisis se puede continuar. En Windows las ventanas de estos archivos se pueden cargar desde el menú `File -> load workspace and File -> load history`. En UNIX, GNU/Linux y Mac OS X se pueden cargar los archivos con el siguiente código:

```
> load(".RData")
```

Ayuda en R

R viene con archivos de ayuda integrados. Se puede acceder directamente desde R o usando un navegador web. El navegador web es especialmente adecuado para la búsqueda de posibles comandos y ejemplos de utilización. El acceso directo desde el entorno es más rápido y se hace con la siguiente instrucción:

```
> help.start()
```

Esto abre una nueva ventana del navegador con los ítems siguientes: una introducción a R, conceptos básicos, manuales y paquetes. También aparece una descripción detallada de comandos y conjuntos de datos en cada paquete.

Por otro lado, dispone de un motor de búsqueda que puede ser utilizado para buscar comandos y conjuntos de datos.

Para acceder directamente desde R a la ayuda de comandos, el nombre del comando debe escribirse entre paréntesis:

```
> help(sum)
```

Al ejecutar este comando, se abre la página de ayuda en R. Por lo general en Windows se abre una nueva ventana para la página de ayuda. En UNIX, en cambio, la página de ayuda se muestra en el editor.

La disposición general de la página de ayuda es la siguiente: la primera línea da el nombre del comando y el paquete donde se puede encontrar. En segundo lugar, encontraremos el nombre de la función o el comando. Después del nombre viene una descripción breve y general sobre el uso del comando. Por último se especificarán los argumentos que puede llevar asociado el comando. Al final de la página de ayuda se encuentran algunos ejemplos de aplicación de la función.

Asignación de datos

La asignación se realiza a través del operador "<-" que se compone del signo menor unido al signo menos. La asignación se utiliza para guardar información en algún objeto llamado en la memoria R. La asignación elimina el objeto que tenga el mismo nombre.

Para guardar un solo número de un objeto denominado `number` se hace de la siguiente manera:

```
> number<-2
```

Los nombres de los objetos tienen unas normas. No pueden empezar con un número y no puede contener caracteres especiales, tales como ä, ñ, etc. Si el nombre de la variable es largo, es preferible separar las partes con un punto (.), no con espacios en blanco (), un signo menos (-) o un guión bajo (_). También es mejor evitar el uso de nombres de objetos iguales a nombres de instrucciones.

2.2.2. Tipos de objetos en R

A continuación se presentan los diferentes tipos de objetos que encontramos en el programa R.

Es muy importante conocer el tipo de objeto con el que estamos trabajando de cara a usar las instrucciones, ya que no todas las instrucciones y funciones están disponibles para todos los objetos. Algunas instrucciones estadísticas pueden generar listas muy largas de objetos.

Para obtener una lista de todos los elementos contenidos en el objeto podemos utilizar `str()` de la forma siguiente:

```
> dat3<-matrix(c(1,2,3,11,12,13),nrow=2,ncol=3,dimnames=list(c(),
+ c("col1","col2","col3")))
> class(dat3)

[1] "matrix"

> str(dat3)
num [1:2, 1:3] 1 2 3 11 12 13
- attr(*, "dimnames")=List of 2
..$ : NULL
..$ : chr [1:3] "col1" "col2" "col3"
```

En este ejemplo el comando `str()` ofrece, en una primera línea, una descripción detallada de una matriz que contiene 3 columnas y 2 filas. En la segunda línea dice que hay nombres para las dimensiones de la matriz, y que estos nombres configuran una lista. Las columnas de la matriz tienen los nombres col1, col2 y col3, pero las filas no contienen los nombres y por consiguiente los nombres aparecen como NULL (sin datos).

Vectores

Un vector es una lista ordenada de números o cadenas. La forma de definirlo es mediante `c()` indicando entre los paréntesis los elementos del vector separados por comas.

Por ejemplo:

```
> x <- c(1, 2, 3); y <- c("a", "b", "Hola")
```

`x` es un vector con tres valores numéricos e `y` un vector con tres cadenas de caracteres.

```
> z1 <- c(TRUE, TRUE, FALSE)
```

`z` es un vector con tres valores lógicos (cierto, cierto y falso).

Una de las funciones útiles al trabajar con vectores es `lenght()`, que calcula la longitud (número de valores) de un vector:

```
> a<-c(2.258, 3.458, 4.897, 1.241, 2.741)
> length(a)
[1] 5
```

Factor

Un factor es un vector de elementos categóricos (o categorías) que suelen ser utilizados cuando es preciso definir grupos, por ejemplo en los modelos de análisis de la varianza.

Un vector cuyos elementos sean números o cadenas puede convertirse fácilmente en un factor utilizando el comando `as.factor()`:

```
> xf<-as.factor(c(1,1,1,2,2,2,3,3,3))
> xf

[1] 1 1 1 2 2 2 3 3 3
Levels: 1 2 3
```

Por defecto, `as.factor()` convierte el vector a un factor con tantos niveles distintos como valores distintos estaban presentes en el vector.

Matriz

Una matriz es una tabla de `n` filas y `m` columnas. Todas las columnas de una matriz deben contener información del mismo tipo. Por lo tanto, podemos tener matrices numéricas, o con datos del tipo cadena, pero no se pueden mezclar en la misma matriz los números y las cadenas.

Por ejemplo:

```
> dat2<-matrix(c(0,1,1,1,0,0,1,1,1,1,0,0),
+ nrow=6,ncol=2,dimnames=list(c(),c("x","y")))
> dat2
   x  y
[1,] 0 1
[2,] 1 1
[3,] 1 1
[4,] 1 1
[5,] 0 0
[6,] 0 0
```

Podemos acceder a todos los valores de una matriz mediante un subíndice. El subíndice indica la fila y la columna de la observación a la que queremos tener acceso dentro de la matriz:

```
> # La primera fila y primera columna
> dat2[1,1]

x
0

> # Solo la primera fila
> dat2[1,]

x y
0 1

> # Solo la primera columna
> dat2[,1]

[1] 0 1 1 1 0 0

> # Las filas de 1 a 3
> dat2[1:3,]

     x y
[1,] 0 1
```

```
[2,] 1 1
[3,] 1 1
```

Las dimensiones de una matriz se pueden comprobar con el comando `dim()` que imprime dos números. El primero indica el número de filas y el segundo el número de columnas:

```
> dim(dat2)


[1] 6 2
```

Data frame

Un `data frame` es una matriz, donde las diferentes columnas pueden contener información de diferentes tipos.

En contraste con las matrices, un `data frame` puede contener columnas de números junto a columnas de cadenas. Es el objeto que se utiliza para representar una matriz de datos donde las columnas son las variables y en las filas se encuentran los individuos o elementos sobre los que se han medido las variables. En un `data frame` las columnas tienen un nombre o etiqueta (el de la variable) y también se les puede dar un nombre a las filas (identificador del elemento). La manera de acceder a un valor concreto se puede hacer de forma idéntica a como se hace con las matrices. Los nombres de las columnas y la filas se pueden encontrar usando las instrucciones `colnames()` y `rownames()`, respectivamente.

Podemos crear un `data frame` a partir de 2 vectores:

```
> x<-c(0,1,0,0,1,0,0,1,0)
> y<-c(0,0,1,0,0,1,0,0,1)
> dat2<-data.frame(x, y)
```

Los nombres de los valores del `data frame` son muy fáciles de cambiar. Por ejemplo:

```
> dat2


  x y
1 0 0
2 1 0
3 0 1
4 0 0
5 1 0
6 0 1
7 0 0
```

```
8 1 0
9 0 1

> names(dat2)

[1] "x" "y"

> row.names(dat2)

[1] "1" "2" "3" "4" "5" "6" "7" "8" "9"

> # Para nombrar cada fila usamos letras
> l<-letters[1:9]
> l

[1] "a" "b" "c" "d" "e" "f" "g" "h" "i"

> row.names(dat2)<-l
> row.names(dat2)

[1] "a" "b" "c" "d" "e" "f" "g" "h" "i"
```

Además de los subíndices, podemos acceder a cada columna de un `data frame` usando su nombre. El nombre se escribe después del nombre del `data frame` seguido de un signo de dólar (`$`):

```
> dat2$x
[1] 0 1 0 0 1 0 0 1 0
```

Una matriz existente se puede convertir en un `data frame` con un comando de conversión específico como:

```
> dat2 <- as.data.frame (dat2)
```

Lista

Una lista es un vector generalizado, es decir, un contenedor diseñado para almacenar objetos de tipos potencialmente distintos (y, por lo tanto, pueden ser otras listas).

```
> una.lista <- list(un.vector = 1:10, # primer elemento un vector
+ una.palabra = "Hola", # segundo: cadena de caracteres
+ una.matriz = matrix(rnorm(20), ncol = 5),
+ # el tercer elemento: es una matriz
+ otra.lista = list(a = 5, b = factor(c("a", "b")))
```

```
+ ) # cuarto: lista de 2 elementos
```

Instrucciones de R

Los comentarios se escriben a continuación del símbolo #. Si el comentario ocupa más de una línea, en cada nueva línea se debe escribir el símbolo que lo caracteriza. Las instrucciones de R se escriben todas en minúscula o en mayúscula y llevan un paréntesis inmediatamente después del nombre. Por ejemplo, consideremos la instrucción `mean()` que permite calcular la media aritmética de un objeto (x). Por ejemplo, la edad de diez personas se pueden almacenar en una variable que llamaremos edad. Si queremos buscar la media de edad del grupo, se puede calcular con:

```
> mean(edad)
```

Cada instrucción se compone de tres partes:

- nombre de la instrucción
- paréntesis
- argumento(s) que irá dentro del paréntesis

No siempre es fácil adivinar qué argumentos tiene cada instrucción. Si queremos conocer los argumentos de nuestra función, lo mejor es consultar la página de ayuda de nuestra instrucción en concreto. Además, buscar las instrucciones en la página de ayuda es muy útil para probar los ejemplos.

En general, los valores numéricos o lógicos se escriben sin comillas. En el caso de los valores en las cadenas de carácter, estos se deben escribir entre comillas. A continuación se presenta un ejemplo del comando `mean()` (ver la ayuda de la función para conocer el significado de cada uno de los argumentos):

```
> mean(age, na.rm=T, trim=0.1)
```

Las instrucciones se pueden escribir una tras otra en la misma línea (anidadas), siempre y cuando el número de paréntesis sea el correcto. Por ejemplo, si queremos quitar los valores perdidos –a los que hemos asignado como valor la cadena “NA”, que quiere decir *not available,* de un conjunto de edades para poder calcular la media– podemos escribir:

```
> # Elimina los valores perdidos de "age" y crea age2
> age2<- na.omit(age)
> # Calcula la media de "age2" (no missing values)
> mean(age2)
> # El mismo proceso pero en la misma línea
> mean(na.omit(age))
```

El entorno de R

El entorno de R tiene algunas funciones útiles. Se pueden ver instrucciones anteriores con las teclas de flecha arriba y abajo. Esta funcionalidad es especialmente útil cuando hay un comando que se ha escrito de forma incorrecta. Una vez se usa la tecla hacia arriba, vuelve a la última orden, que puede ser editada sin necesidad de volver a escribir todo el comando. También es posible obtener una lista de todas las instrucciones ejecutadas en R utilizando el historial de instrucciones `history()`. Por defecto R proporciona una lista con un par de docenas de instrucciones. Para ampliar la lista, se puede usar el argumento max.show. Por ejemplo, el comando siguiente muestra las últimas 100 instrucciones:

```
> history(max.show=100)
```

Los paquetes en R

En R se pueden instalar los paquetes adicionales tal y como se describe en el subapartado de instalación detallado más arriba. Los paquetes instalados no se cargan en R por defecto cada vez que se inicia. Es el usuario quien debe cargar el paquete en la memoria utilizando la instrucción `library()`. Por ejemplo, para cargar el paquete `(rpart)` la instrucción será:

```
> library(rpart)
```

Para obtener una lista de todos los paquetes cargados en la memoria, se utiliza `sessionInfo()`:

```
> sessionInfo()
```

Si el paquete no se carga en la memoria, las instrucciones o conjuntos de datos contenidos en el paquete no pueden ser utilizados y no se encontrará documentación para implementar las funciones. Por ejemplo, la ayuda `rpart()` daría el siguiente mensaje de error:

```
No documentation for 'rpart' in specified packages and libraries:
you could try 'help.search("rpart")'
```

Y esto significa que no tiene información sobre la función `rpart` en el paquete especificado o en la biblioteca y pedirá que el usuario busque la función en la ayuda de la biblioteca.

A veces dos paquetes contienen funciones con exactamente el mismo nombre. Esto puede provocar la advertencia siguiente:

```
Attaching package: 'rpart'
The following object(s) are masked \_by\_ .GlobalEnv :
```

```
rpart
```

Esta advertencia significa simplemente que el comando `rpart()` no se puede utilizar, porque ya hay otro comando con el mismo nombre en la memoria. Esto puede solucionarse eliminando el paquete conflictivo de la memoria mediante el comando `detach()`:

```
> detach(package:rpart)
```

Cambiar el directorio de trabajo

Es una buena práctica dedicar una sola carpeta para contener todos los ficheros relacionados con un único conjunto de datos. Cuando se empieza a trabajar con un nuevo conjunto de datos, se deben copiar en una carpeta nueva. Antes de la importación de datos, R debe decidir dónde residirán estos. En Windows podemos escoger el directorio de trabajo usando el menú `File -> Change Dir`. Esto abre una ventana nueva donde el usuario puede escoger la carpeta correcta. En UNIX, GNU/Linux y Mac OSX, el usuario debe especificar la ruta de la carpeta de datos con el comando `setwd()`. Por ejemplo, para establecer la ruta a una carpeta de datos, el usuario puede ejecutar un comando de esta forma:

```
> setwd("/home/uoc/doc/timeseries")
```

A veces es útil para verificar el directorio de trabajo actual. El camino hacia el directorio de trabajo se puede comprobar con el comando `getwd()`:

```
> getwd()
```

Los archivos que residen en la carpeta actual se pueden listar usando el siguiente comando `dir()`:

```
> dir()
```

2.2.3. Uso de R como calculadora

R contiene una colección muy completa de funciones matemáticas y aritméticas. En R se utiliza el orden estándar de precedencia en las operaciones, por lo cual se calcularán primero las potencias y las raíces, seguido de la multiplicación y la división y, por último, las sumas y las restas.

Operaciones aritméticas

Las sumas y restas se hacen utilizando los operadores + y -:

```
> 8+3
[1] 11
```

```
> 8-3
[1] 5
```

La multiplicación se realiza mediante un asterisco (*):

```
> 4*6
[1] 24
```

La división utiliza la barra (/):

```
> 16/4
[1] 4
```

Las potencias usan el acento circunflejo ^:

```
> 3^3
[1] 27
```

Para la raíz cuadrada, se usa el comando `sqrt`:

```
> sqrt(9)
[1] 3
```

Funciones matemáticas

Además de las funciones aritméticas, en R podemos encontrar toda clase de funciones (instrucciones) matemáticas disponibles. Las funciones más importantes son los logaritmos y exponentes.

Los logaritmos se calculan utilizando el comando `log()`:

```
> log(4)
[1] 1.386294
```

El comando `log()` es el logaritmo en base 2.718282, es decir, el logaritmo neperiano. Los logaritmos en base 10 se pueden calcular utilizando la función `log10()` y el logaritmo en base 2 con `log2()`. El comando `exp()` realiza la función inversa al logaritmo, es decir, calcula el valor de *e* elevado al valor numérico que se indica en la función:

```
> exp(2)
[1] 7.389056
```

Funciones lógicas

Las pruebas lógicas para comparar la igualdad de dos objetos se pueden hacer usando el operador ==, y el resultado es una expresión lógica que puede ser verdadera (`TRUE`) o falsa (`FALSE`):

```
> 8==8
[1] TRUE
```

Otros posibles operadores son >, <, >=, <= y != que corresponden a mayor que, menor que, mayor o igual, menor o igual o desigual. Por ejemplo, al evaluar las expresiones 3 < 5 y 3 > 5 se obtendrá `TRUE` y `FALSE` respectivamente.

```
> 3<5
[1] TRUE
> 3>5
[1] FALSE
```

Número de decimales

El programa R realiza los cálculos con más decimales de los que se muestran en la salida. Por lo tanto, el número de decimales en la salida se puede limitar especificando el número de decimales que queremos con el comando `round()`:

```
> round(exp(2), digits=2)
[1] 7.39
```

2.2.4. Entrada y salida de datos

En R podemos introducir los datos de varias maneras. Para la entrada de datos, la forma más sencilla es manualmente utilizando las funciones proporcionadas por el mismo programa. Otra menos laboriosa y que sirve para grandes conjuntos de datos es leer los datos de una tabla que se encuentra en un fichero externo. La tabla se puede crear en cualquier editor de hojas de cálculo, como Microsoft Excel, y guardar en un formato de texto delimitado por espacios o tabulaciones (ficheros .txt) o separado por comas (ficheros .csv). Por defecto, en R el delimitador de decimales para los valores numéricos es el punto (.). La coma (,) es un delimitador de las listas. Por lo tanto, todos los valores numéricos que se importan en R deben ser evaluados por si contienen solo números o números y puntos. De lo contrario los valores se leen en forma de texto.

Escritura de datos tabulares

Los datos tabulares se pueden escribir en un archivo usando el comando `write.table()`. El comando toma como argumentos el nombre del objeto a ser escrito, el nombre del archivo entre comillas y el tipo de separador de archivo (en este caso, tab). Por ejemplo:

```
> # Creamos un data.frame
> nombres <- sample(LETTERS, 10)
> numeros <- runif(10)
> ab <- data.frame(nombres, numeros)
> write.table(ab, file="ab.txt", sep="\t")
```

Lectura de datos tabulares

Los datos tabulares, tales como los archivos de texto delimitado por tabuladores, se pueden leer fácilmente en R mediante el comando `read.table()`. Se debe introducir el nombre del archivo (entre comillas), un valor lógico que indica si las columnas de la tabla tienen títulos, el separador utilizado en el archivo y, opcionalmente, el número de la columna con los nombres de las filas:

```
> dat<-read.table("ab.txt", header=TRUE, sep="\t", row.names=1)
> dat
```

En este caso, un archivo de texto delimitado por tabuladores que contiene dos columnas denominadas *a* y *b* se leerá en un objeto dat. Si el separador decimal es diferente al punto (.), se puede especificar en el comando `read.table()` con el argumento `dec`. Por ejemplo, el siguiente comando leería un archivo de texto con los decimales separados por `","`:

```
> dat<-read.table("ab.txt", header=TRUE, sep="\t", row.names=1,
+ dec=",")
```

Guardar el resultado en un archivo

Las salidas generadas por instrucciones de R se pueden guardar en un archivo utilizando la función `sink()` indicándole como argumento, el nombre de un archivo para guardar el resultado. Después de ejecutar las instrucciones, si escribimos de nuevo `sink()` ya no se guardará ningún resultado más. Por ejemplo:

```
> sink("result.txt")
> print(ab)
> print(summary(ab))
> sink()
```

2.2.5. Operaciones con vectores

Todas las instrucciones comentadas anteriormente son aplicables a los vectores. Cuando se ejecuta un comando sobre un vector, la función se aplica en cada elemento del vector de forma individual. También podemos encontrar instrucciones que se usan solo para los vectores.

Operadores básicos con vectores

Podemos usar los operadores básicos para realizar cálculos con vectores:

```
> x<-c(3,1,2,0,1,3,3,1,0)
> y<-c(4,0,1,0,3,2,5,3,2)
> z<-c(0,0,1,0,1,2,0,1,2)
> x;y

[1] 3 1 2 0 1 3 3 1 0
[1] 4 0 1 0 3 2 5 3 2

> x+y

[1] 7 1 3 0 4 5 8 4 2
```

Operadores matemáticos en los vectores

Las instrucciones más utilizadas para los vectores con elementos numéricos son sum() y mean(), que calculan una suma o una media aritmética de los valores del vector:

```
> sum(x)
[1] 14
> mean(x)
[1] 1.555556
```

Las instrucciones min(), max() y range() nos calculan el mínimo, el máximo y el rango de los números en un vector:

```
> min(x)
[1] 0
> max(x)
[1] 3
> range(x)
[1] 0 3
```

Podemos obtener el resumen de las estadísticas básicas de un vector de datos con:

```
> summary(x)

Min. 1st Qu. Median Mean 3rd Qu. Max.
0.000 1.000 1.000 1.556 3.000 3.000

> summary(y)
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.000 1.000 2.000 2.222 3.000 5.000
```

Para el cálculo de la desviación estándar, la varianza y la correlación, podemos usar:

```
> sd(x)
[1] 1.236033
> var(x)
[1] 1.527778
> cor(x,y)
[1] 0.5828083
```

2.2.6. Manipulación de los datos

Generación de secuencias numéricas

La forma más sencilla de generar una secuencia de números es mediante el uso de los dos puntos (:). Cuando dos puntos separan dos números se genera una secuencia de números. Por ejemplo:

```
> 1:6
[1] 1 2 3 4 5 6
```

Para generar secuencias más complicadas podemos usar el comando `seq()`. Este comando necesita tres argumentos. El primero indica el comienzo de la secuencia, el segundo el final de la secuencia, y el tercero, el incremento o decremento a usar. Por ejemplo:

```
> seq(1, 6, 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
> seq(6, 1, -0.5)
[1] 6.0 5.5 5.0 4.5 4.0 3.5 3.0 2.5 2.0 1.5 1.0
```

Generar secuencias repetidas

Una secuencia de repetición se puede generar con facilidad con el comando rep(). Una secuencia de repetición es una manera fácil de generar nuevas variables para modelos estadísticos. La función rep() usa dos argumentos, qué valor o valores queremos que se repita y cuántas veces se debe repetir:

```
> rep(1, 5) # Repetir 5 veces el número 1
[1] 1 1 1 1 1
> rep("UOC", 5) # Repetir 5 veces 1a cadena UOC
[1] "UOC" "UOC" "UOC" "UOC" "UOC"
> rep(1:3, 2) # Repetir los números del 1 al 3, dos veces
[1] 1 2 3 1 2 3
```

```
> rep(1:3, each=2) # Repetir los números del 1 al 3, cada uno dos veces
[1] 1 1 2 2 3 3
```

Recodificar en R

La manera más sencilla de recodificar en R es utilizando el comando siguiente `ifelse()`. Esta función tiene tres argumentos, una comparación lógica, el valor a devolver si la comparación es cierta y el valor a devolver si la comparación es falsa.

```
> m<-c(0,1,0,11,0,11,0,0,1)
> # Si es igual a 1, devuelve 1,
> # en otro caso devuelve 0
> x<-ifelse(m==1, 1, 0)
> # Si es igual a 11, devuelve 1,
> # en otro caso devuelve 0
> y<-ifelse(m==11, 1, 0)
> x; y

[1] 0 1 0 0 0 0 0 0 1

[1] 0 0 0 1 0 1 0 0 0
```

Es una buena práctica, para comprobar que nuestro programa funciona y hemos recodificado correctamente, hacer uso del comando `table()`, que calcula una tabla de contingencia de dos vectores:

```
> table(x, y)

  y
x   0 1
  0 5 2
  1 2 0
```

Combinación de tablas

La fusión de dos tablas se hace con el comando `merge()`. Este comando tiene dos argumentos obligatorios, los nombres de las tablas. A menudo es necesario especificar la columna por la que queremos combinar las tablas. Las columnas se especifican mediante opciones `by.x` y `by.y`. Por ejemplo, la fusión de dos tablas podría proceder de la siguiente manera:

```
> p<-data.frame (y=1:5)
> # Nombramos las filas con algunas letras
> row.names(p)<-c(letters[1:2], letters[7:9])
> p
```

```
  y
a 1
b 2
g 3
h 4
i 5

> r<-data.frame(x=6:10)
> # Nombramos las filas con algunas letras
> row.names(r)<-letters[1:5]
> r

  x
a 6
b 7
c 8
d 9
e 10

> # La combinación se hace usando el nombre de las filas
> merge(p, r, by.x="row.names", by.y="row.names")

  Row.names y x
1         a 1 6
2         b 2 7
```

Trasposición de matrices

En ocasiones conviene utilizar la matriz de datos de forma que las columnas se convierten en filas y las filas se convierten en columnas (transposición). Esto resulta especialmente útil cuando determinados modelos estadísticos se aplican a datos de microarrays. Las herramientas estadísticas clásicas asumen que las columnas contienen las variables y las filas las observaciones individuales para las variables. En los datos de microarrays esto no suele ser así, porque los chips individuales constituyen las columnas. La trasposición se realiza mediante el comando `t()`:

```
> # tabla simple de dos vectores
> dat2<-cbind(x,y)
> dat2
     x y
[1,] 0 0
[2,] 1 0
[3,] 0 0
[4,] 0 1
[5,] 0 0
[6,] 0 1
```

```
[7,] 0 0
[8,] 0 0
[9,] 1 0

> dat3<-t(dat2)
> dat3

  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
x   0    1    0    0    0    0    0    0    1
y   0    0    0    1    0    1    0    0    0
```

Clasificar y ordenar

Clasificar y ordenar vectores es una de las operaciones básicas más útiles. Ordenar un vector, ya sea en orden ascendente o descendente, se realiza con el comando `sort()`. El comando `order()` devuelve un vector de permutación que ordenará el vector original en orden ascendente. Un ejemplo de esto es:

```
> i<-data.frame(x=round(abs(c(rnorm(5)*10))), y=c(1:5))
> # Nombrar filas con letras
> row.names(i)<-letters[1:5]
> i

  x y
a 1 1
b 4 2
c 26 3
d 1 4
e 12 5

> # Devuelve los valores en orden ascendente
> sort(i$x)

[1] 1 1 4 12 26

> order(i$x)

[1] 1 4 2 5 3
```

El resultado del comando `order()` es un poco más difícil de entender. Indica que el vector quedará ordenado de forma ascendente tomando los valores que estén en las posiciones 1,4,2,5,3. El comando `order()` se puede utilizar para ordenar tablas completas. Para ordenarlas se utilizan subíndices de la siguiente manera:

```
> # Ordenar la tabla i de acuerdo con x
> i[order(i$x),]
```

```
  x y
a 1 1
d 1 4
b 4 2
e 12 5
c 26 3
```

Ordenar una tabla requiere ser prudente. Como el comando `order()` genera una permutación del vector, las filas de la tabla original se muestran en el orden indicado por el vector de permutación, y se genera la nueva tabla. Esta nueva tabla se ordena de forma ascendente de acuerdo a los valores del vector especificados en el comando `order()`.

Los valores perdidos

Los datos biológicos a menudo contienen valores perdidos. Estos se indican en R con la cadena de texto NA. Los valores perdidos complican muchos análisis, como los clústeres jerárquicos o incluso el cálculo de la media aritmética de un vector.

Hay dos soluciones sencillas para el problema de valores perdidos: la eliminación y la imputación. Si los valores perdidos se eliminan de los datos, todas las filas (observaciones) que contienen al menos un valor perdido serán eliminadas del conjunto de datos. La eliminación se realiza mediante el comando `na.omit()`:

```
> # Creación un vector con un valor perdido
> vwm<-c(1,2,3,NA,5)
> # Eliminación de valores perdidos
> vwom<-na.omit(vwm)
> vwom

[1] 1 2 3 5
attr(,"na.action")
[1] 4
attr(,"class")
[1] "omit"

> # Cálculo de una media del vector
> # Si existen valores perdidos, esto da como resultado NA
> mean(vwm)

[1] NA

> # Sin valores perdidos se obtiene la media
> mean(vwom)
```

```
[1] 2.75
```

El comando `na.omit()` funciona de manera similar para las matrices y para los data frames, pero en lugar de valores individuales, se eliminan las filas enteras.

Los valores perdidos también se pueden sustituir imputando un valor para todas las observaciones que faltan. La función impute() está disponible en la librería `e1071`:

```
> if (!(require(e1071))) install.packages("e1071")
> library(e1071)
```

Hay dos maneras de imputación. Los valores perdidos pueden ser reemplazados, ya sea con la media o con la mediana de otras observaciones en la misma variable. Tened en cuenta que la imputación requiere una matriz o un data frame como entrada.

```
> # Creando una matriz
> v<-matrix(c(1,2,3,4,11,16:20), ncol = 2)
> v

     [,1] [,2]
[1,]    1   16
[2,]    2   17
[3,]    3   18
[4,]    4   19
[5,]   11   20

> v[1,1]<-NA
> v2<-impute(v, what=c("mean"))
> v2
     [,1] [,2]
[1,]    5   16
[2,]    2   17
[3,]    3   18
[4,]    4   19
[5,]   11   20
> v2<-impute(v, what=c("median"))
> v2

     [,1] [,2]
[1,] 3.5    16
[2,] 2.0    17
[3,] 3.0    18
[4,] 4.0    19
```

```
[5,] 11.0   20
```

2.2.7. Bucles y funciones condicionales

Los bucles son estructuras de control que permiten uno o varios comandos en repetidas ocasiones. Si hay más de un comando a ejecutar, el conjunto de comandos debe llevar una llave (`{`) al inicio y otra (`}`) al final. La instrucción más habitual para generar bucles que se ejecutan un número fijo de veces es la que utiliza la instrucción `for()`.

Las funciones condicionales permiten ejecutar un comando o grupo de comandos solo si se cumple una determinada condición descrita por el usuario. La ejecución condicional de R se genera con las instrucciones `if()` y `if()...else`.

for

El bucle for requiere un índice y el número de veces que queremos repetir la misma orden. El índice puede ser cualquier objeto de R, aunque lo mejor es evitar los nombres de objetos ya existentes. Las líneas que pertenecen al bucle se escriben entre llaves. Por ejemplo, un bucle que calcula 2^x para valores de x que van desde 1 hasta 5, se escribe como:

```
> # En este ejemplo i es el índice
> # que se repite 5 veces
> for(i in 1:5) {print(2^i)}


[1] 2
[1] 4
[1] 8
[1] 16
[1] 32
```

if

La instrucción `if()` contiene entre los paréntesis una operación lógica. Si la evaluación de la comparación lógica da como resultados que es verdadera, las instrucciones que van entre llaves se ejecutan. En el ejemplo anterior, si solo se desea realizar el cálculo de 2^x si una variable *k* es igual a 1, el código será:

```
> k<-1
> if (k ==1) {for (i in 1:5) {print (2^i)}}


[1] 2
[1] 4
[1] 8
[1] 16
```

```
[1] 32
```

Se puede probar la ejecución condicional en el ejemplo anterior, cambiando el valor de *k* para, por ejemplo: 2, 3...

if... else

A veces nos conviene ejecutar uno o varios comandos si la condición es cierta y otra serie distinta si es falsa. Por ejemplo, en el programa siguiente 2^2 se calcula solo si *k* es igual a 1 en caso contrario se calcula 3^2:

```
> k<-1
> if(k==1){print(2^2) } else { print(3^2)}


[1] 4
```

Se puede implementar el mismo procedimiento utilizando dos condiciones creadas con if:

```
> if (k==1) {print(2^2)}


[1] 4


> if (k!=1) {print(3^2)}
```

A veces, necesitamos realizar una operación lógica que contiene varias comparaciones a la vez. Para implementar varias condiciones podemos usar los operadores | y &. El primer operador significa "o" y el segundo, "y". Por ejemplo, en el caso anterior, si se desea calcular 2^2 cuando $k = 1$ o 2, y 3^2 en los otros casos, el código sería:

```
> if(k==1 | k==2) {print (2^2)}


[1] 4


> if(k> 2) {print (3^2)}
```

2.2.8. Gráficos en R

El lenguaje R tiene capacidad de generar gráficos de muy buena calidad.

El comando plot

`plot()` es el comando general para crear gráficos de primer orden. Se puede utilizar para visualizar diferentes tipos de objetos. La imagen producida por este comando es típicamente un diagrama de dispersión de dos vectores:

```
> # Generamos dos vectores aleatorios
> # con 100 valores de una distribución normal
> x<-rnorm(100)
> y<-rnorm(100)
> # Representación de los valores en un gráfico de dispersión (scatterplot)
> plot(x, y)
```

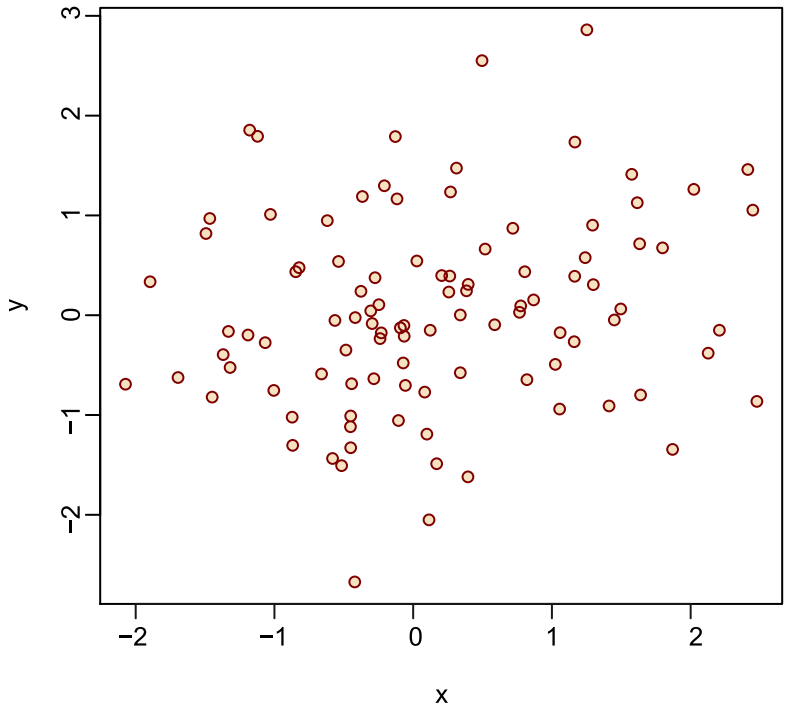

Figura 10. Representación gráfica de un Scatterplot
Un diagrama de dispersión sirve para mostrar los valores de dos variables cuantitativas relacionadas

Si se aplica la función `plot()` a una tabla (matriz o data frame), se se obtiene una matriz de gráficos de las variables 2 a 2. Por ejemplo:

```
> # Creación de un data frame con cuatro columnas
> # llamadas a, b, c, y d
> dat <-data.frame (a=rnorm(100), b=rnorm(100), c=rnorm(100), d=rnorm (100))
> plot(dat)
```

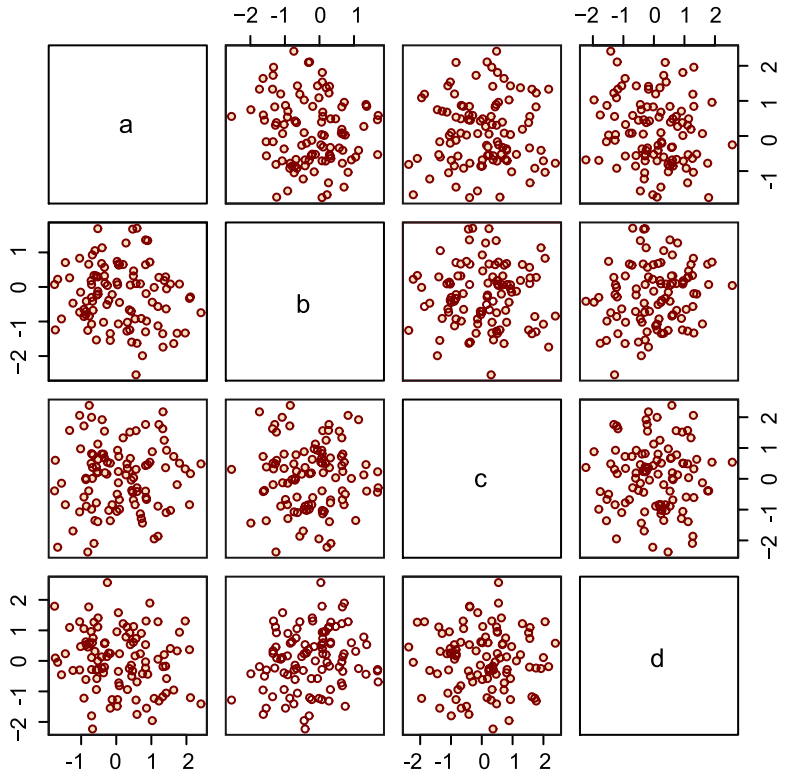

Figura 11. Representación gráfica de un conjunto de Scatterplots
Scatterplots 2 a 2 de una matriz. Con un conjunto de Scatterplots podemos ver representadas las relaciones 2 a 2 entre diferentes variables.

El comando `plot()` contiene otros argumentos. Para escoger cómo queremos el gráfico, usaremos el argumento `type`. Los tipos posibles son los siguientes puntos (p), líneas (l), puntos y líneas (b), overplotted (o), como las barras de histograma (h), y pasos (s).

También se puede añadir un título y dar nombres a los ejes. Por ejemplo:

```
> # El título principal de un gráfico de dispersión y de x
> # y las etiquetas de eje y
> plot(x, y, main = "Gráfico de dispersión", xlab= "X variable", ylab= "variable Y")
```

Histograma

Los vectores pueden ser visualizados como histogramas. El comando para generar histogramas es `hist()`. Por defecto podemos disponer de 10 barras para la creación de nuestro gráfico, pero podemos cambiar el número de barras con el argumento `breaks()`. Por ejemplo, el código siguiente produce la figura 12.

```
> hist (x)
> hist (x, breaks = 40)
```

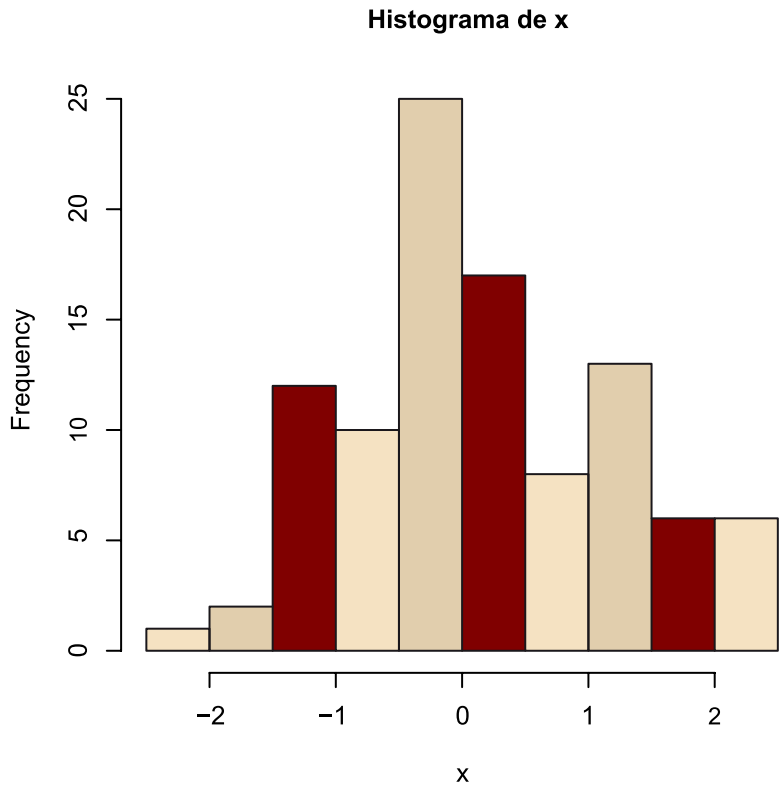

Figura 12. Histograma
Un histograma es la representación grafica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados.

Boxplot

Los boxplots o diagramas de caja se pueden producir usando el comando siguiente `boxplot()`. Este comando usa para representar un único vector como un diagrama de caja única o bien un data frame como varios diagramas en la misma figura. Si se representa una matriz con `boxplot()`, se tratará como un vector, y solo se producirá un único diagrama de caja. Por ejemplo, el código siguiente crea la imagen de la figura 13.

```
> # Hacemos el gráfico de un vector
> boxplot(x)
```

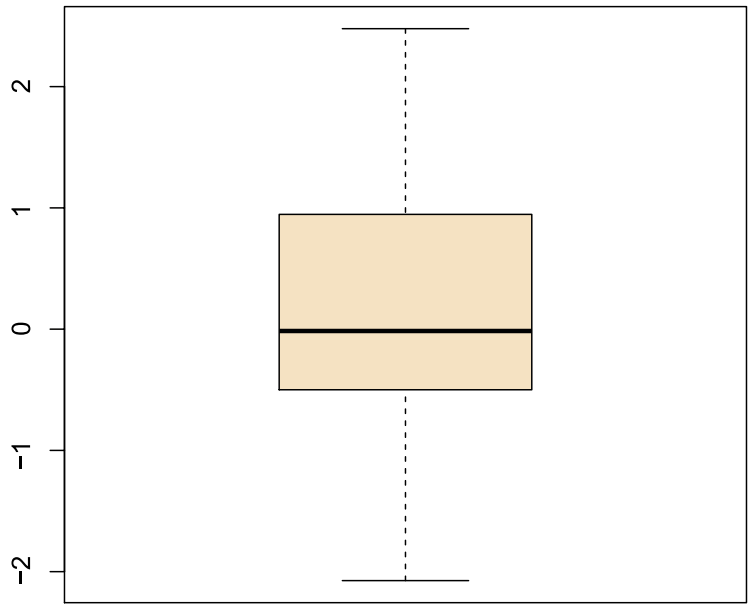

Figura 13. Boxplot
El boxplot es un diagrama con una "caja y bigotes", que representa el mínimo, el máximo y los cuartiles del 25%, 50% y 75% de una variable.

Scatterplot

El diagrama de dispersión o Scatterplot se produce usando dos vectores en el comando `plot()` como ya se ha descrito más arriba.

Panelplots

Los gráficos de panel son figuras en las que se visualizan diferentes gráficos en un mismo marco. Por lo general todos los paneles son del mismo tipo, pero no necesariamente. Por ejemplo, el comando `pairs()` genera un panel de diagramas de dispersión para un conjunto de datos. Para definirlos se utiliza el argumento `mfrow` del comando par, que permite determinar en cuántas filas y columnas se dispondrán los gráficos en el panel. Después de establecer las dimensiones, se genera un número igual de gráficos. Estos se representan en el panel desde la parte superior izquierda a la esquina inferior derecha del panel. Por ejemplo, la figura 14 se produjo utilizando las siguientes instrucciones:

```
> # Las dimensiones del panel:
> # 2 filas y dos columnas
> par (mfrow = c (2,2))
> hist (x, main = "A")
> hist (x, rompe = 20, main = "B")
> boxplot(x, main="C")
> boxplot(dat, notch=T, main="D")
> par(mfrow=c(1,1))
```

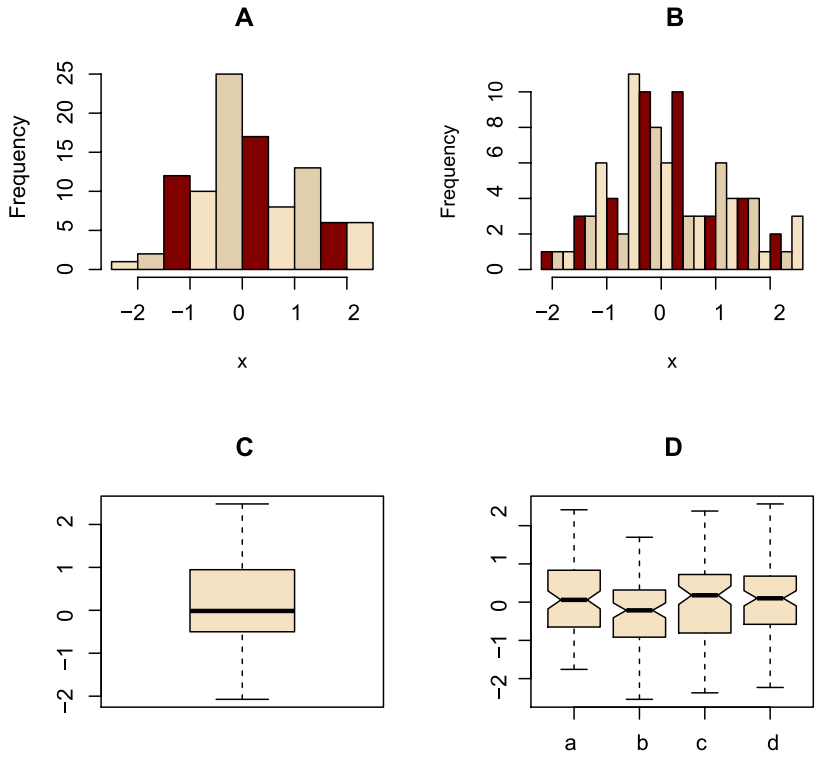

Figura 14. Gráfico de panel con cuatro figuras distintas
Un gráfico de panel permite visualizar simultáneamente varios gráficos del mismo tipo o de tipos distintos.

Otras opciones gráficas

Como ya hemos comentado, la configuración gráfica se puede cambiar con el comando `par()`. A continuación, explicaremos algunas de las opciones más comúnmente utilizadas. Si los ajustes se aplican utilizando el comando `par()`, se aplicarán a todos los gráficos que hemos producido en la misma sesión de R. Si las opciones se aplican con `plotwise`, la configuración se aplica solo al gráfico actual. Una de las tareas más comunes es cambiar el rango de los ejes.

En R este trabajo se realiza mediante los argumentos `xlim` y `ylim`. Ambos argumentos toman un vector que tiene dos valores, el límite inferior y el límite superior del rango del eje. Por ejemplo,

```
> # Intervalo determinado por el rango de los datos
> plot(x, y)
> # Definimos el rango del eje
> plot(x, y, xlim=c(-1, 1), ylim=c(-1, 1))
```

Otra tarea común es cambiar el símbolo y el tamaño de la fuente. Para ello se utiliza `cex-arguments`, que cambia el tamaño del símbolo, `cex.axis`, `cex.label` y `cex.main` que cambian el tamaño de la anotación del eje *x* e *y*, las etiquetas de eje y el título principal.

```
> plot(x, y, cex = 0.5, cex.axis = 0.5, cex.lab = 0.5)
```

Si el tipo de gráfico es de líneas se debe poner (type = "l"), y a continuación indicar el tipo y ancho de la línea con los argumentos `lty` y `lwd`, respectivamente. Por ejemplo:

```
> # Generación de un grafico de líneas discontinuas
> # con un grosor de 2
> z=1:length(x)
> plot (z, x, type = "l", lty = 2, lwd = 2) -2
```

La forma habitual para configurar el tipo de línea son los números 0-6 (0 = blanco, 1 = sólido, 2 = discontinua, 3 = puntos, 4 = puntos y guiones, 5 = guion largo, 6 = doble raya). El ancho de la línea del gráfico puede ser cualquier número positivo, aunque sea un número decimal (aunque no todos los tipos de gráficos soportan anchos de línea inferior a 1).

Incluir objetos en los gráficos

Se pueden añadir determinados objetos a la imagen de un gráfico ya creado. El comando `abline()` añade una línea horizontal o vertical en una posición determinada del gráfico ya existente:

```
> plot (x, y)
> abline (h = 0)
> abline (v = 0)
```

Los argumentos `h` y `v` especifican la posición horizontal o vertical de la línea, aunque puede añadirse cualquier línea, por ejemplo, la recta de regresión entre *x* e *y*.

```
> plot (x, y)
> abline(lm(y~x))
```

Etiquetar los puntos de un gráfico

A veces es interesante conocer las etiquetas de las observaciones de un gráfico de dispersión. Las etiquetas se pueden agregar utilizando el comando `text()`. Se requieren tres argumentos para este comando, la coordenada *x* del texto, la coordenada *y* del texto y el texto a representar:

```
> plot (x, y)
> text (x, y-0.1, z, cex = 0.5)
```

Si hay necesidad de añadir nuevas observaciones al gráfico, podemos usar el comando `points()`. Este comando tiene dos argumentos, `x` e `y` que serán las coordenadas de la nueva observación. El color (s) de la nueva observación (s) se especifica mediante el parámetro `col` y el tamaño y el tipo de símbolo se puede cambiar al mismo tiempo con los argumentos `cex` y `pch`:

```
> plot(x, y)
> points(-2, -2, col="red", cex=1.5, pch=19)
```

Las leyendas siempre se añaden después de realizar el gráfico. Para incluir una leyenda usaremos el comando `legend()`. Los argumentos que se necesita variar con la posición de la leyenda (x) y el texto que se escribe en la leyenda.

```
> # Grafico de dispersión con leyenda
> plot (x, y)
> points(1, -2, col = "red", pch = 19)
> legend ("bottomright", legend=c("Original", "Added later"), fill = c ("black", "red"))
```

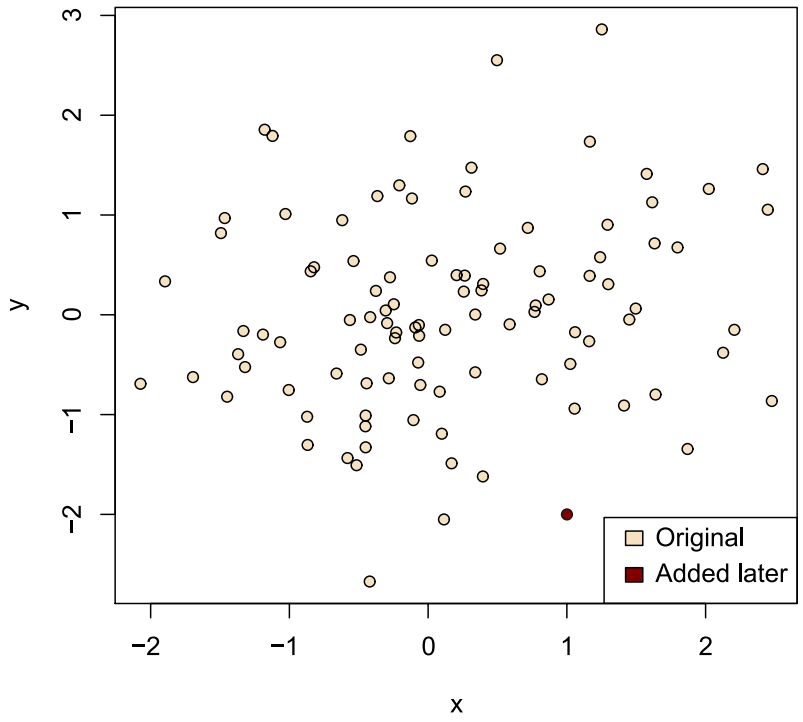

Figura 15. Diagrama de dispersión con puntos personalizados y leyenda

Almacenamiento de imágenes

Existen distintos modos de guardar las imágenes en R. En Windows es posible activar las ventanas de la imagen (que se activa justo después de crear el gráfico), y guardarla utilizando el menú `File->Save as`. Podremos guardar la imagen en formato de metarchivo de Windows (no disponible en otros sistemas), PostScript, PDF, PNG, mapa de bits o JPG.

De forma general, las imágenes también se pueden guardar en un determinado formato usando una instrucción antes de generar el gráfico. El dispositivo de gráficos se activa, el gráfico se genera y el dispositivo se cierra. Los dispositivos gráficos se abren con las instrucciones `PostScript()`, `pdf()`, `png()`, `bmp()`, `jpeg()`. El gráfico se genera como de costumbre, y los dispositivos se cierran con el comando `dev.off()`. Todas las instrucciones del dispositivo gráfico necesitan opciones ligeramente diferentes, así que es recomendable consultar la página de ayuda. Un ejemplo es:

```
> jpeg("scatterplot.jpg")
> plot(x, y)
> dev.off()
```

El gráfico que se genera con el comando `plot(x, y)` se guarda en un archivo scatterplot.jpg.

2.2.9. Operaciones con matrices de datos

A menudo es importante calcular la media o las desviaciones estándar de filas o columnas para extraer conclusiones relevantes en estudios biológicos. Este tipo de cálculos en una matriz de datos se pueden hacer mediante bucles. Sin embargo, es mucho más conveniente usar el comando `apply()`, que está orientado a aplicar de forma rápida una función a todas las filas o columnas de una matriz o data frame. Para ello se especifica el nombre de la matriz (o data frame), un valor que indica si se debe aplicar a las filas o a las columnas (1 para las filas y 2 para columnas), y por último, el nombre de la función. Para ilustrarlo, calculamos el valor medio de las expresiones de los genes de una serie de personas.

```
> gendat<-matrix(c(0.0125, 0.0625, 0.0250, 0.0125, 0.0625, 0.0650,
+ 0.0625, 0.0250, 0.0125, 0.0650, 0.0625, 0.0250),nrow=4,ncol=3,
+ dimnames=list(c("gene1", "gene2", "gene3", "gene4"),c("Ana",
+ "Teresa","Elena")))
> apply(gendat,2,mean)

     Ana   Teresa    Elena
0.028125 0.053750 0.041250
```

Del mismo modo, para calcular la media de cada gen (fila):

```
> apply(gendat,1,mean)
     gene1      gene2      gene3      gene4
0.02916667 0.06416667 0.05000000 0.02083333
```

Ocurre con frecuencia que se desea reordenar las filas de una matriz de acuerdo a un cierto criterio o, más específicamente, los valores de una determinada columna de un vector. Por ejemplo, para reordenar la matriz `gendat` de acuerdo con las medias de la fila, es conveniente guardar los valores en un vector y utilizar el comando `order()`.

```
> meanexprsval <- apply(gendat,1,mean)
> o<-order(meanexprsval,decreasing=TRUE)
> o


[1] 2 3 1 4
```

De este modo `gene2` aparece en primer lugar porque tiene la mayor media (0.06416667), luego viene `gene3` (0.05000000), seguido por `gene1` (0.02916667) y, finalmente, `gene4` (0.02083333). Finalmente, se puede reordenar toda la matriz mediante la especificación o en el índice de la fila.

```
> gendat[o,]
          Ana Teresa  Elena
gene2 0.0625 0.0650 0.0650
gene3 0.0250 0.0625 0.0625
gene1 0.0125 0.0625 0.0125
gene4 0.0125 0.0250 0.0250
```

Para la selección de elementos de la matriz o data frame con una cierta propiedad, tenemos varias opciones:

1) Indicando en un vector los números de las filas o columnas a seleccionar

```
> gendat[c(1,2),]
          Ana Teresa  Elena
gene1 0.0125 0.0625 0.0125
gene2 0.0625 0.0650 0.0650
```

2) Indicando en un vector los nombres de las filas o columnas a seleccionar

```
> gendat[c("gene1","gene2"),]
          Ana Teresa  Elena
gene1 0.0125 0.0625 0.0125
gene2 0.0625 0.0650 0.0650
```

3) Una tercera y más avanzada forma de hacerlo es proceder a una evaluación en términos de `TRUE` o `FALSE` de los elementos lógicos de un vector. Por ejemplo, podemos evaluar si la media de la fila es positiva.

```
> meanexprsval > 0
gene1 gene2 gene3 gene4
```

```
  TRUE  TRUE  TRUE  TRUE
```

Ahora podemos usar la evaluación de `meanexprsval> 0` en términos de los valores `TRUE` o `FALSE` como un índice de la fila.

```
> gendat[meanexprsval > 0,]
         Ana Teresa  Elena
gene1 0.0125 0.0625 0.0125
gene2 0.0625 0.0650 0.0650
gene3 0.0250 0.0625 0.0625
gene4 0.0125 0.0250 0.0250
```

Observad que esto selecciona los genes cuya evaluación es `TRUE`.

2.2.10. Utilización de *scripts*

Es aconsejable usar un editor de texto como el Notepad, Kate, Emacs o RStudio para crear un fichero (*script*) con las instrucciones de R que se van a ejecutar en líneas separadas. Este fichero debe tener extensión `.r`. Para ejecutarlo se puede proceder de formas distintas. Una opción es abrirlo con un editor de texto estándar y copiar y pegar en la ventana de R las instrucciones a ejecutar. Otra posibilidad es abrir el fichero desde R (Menú:File->Open script) y ejecutarlo mediante el botón *Run line or selection*.

2.3. El proyecto Bioconductor

Bioconductor es un proyecto de software libre (código y desarrollo abiertos), basado en el lenguaje de programación R, que proporciona herramientas para el análisis de datos genómicos de alto rendimiento.

El proyecto se inició en el año 2001 e incluye más de 25 desarrolladores entre Estados Unidos, Europa y Australia. Incluye un conjunto de paquetes que presentan diferentes funcionalidades, como análisis de microarrays, secuenciación y detección de SNPs, entre muchas otras.

2.3.1. Objetivos de Bioconductor

Los objetivos de Bioconductor son:

- Proporcionar acceso a potentes métodos estadísticos y gráficos para el análisis de datos genómicos.

- Facilitar la integración de metadatos biológicos (*GenBank*, *GO*, *LocusLink*, *PubMed*) en el análisis de datos experimentales.

- Permitir el desarrollo rápido de software extensible, interoperable y escalable.

- Promover la documentación de alta calidad y la investigación reproducible.

- Proporcionar formación en métodos de cálculo y estadística.

2.3.2. Paquetes

Para diseñar y distribuir el software de Bioconductor se usa el lenguaje R y su sistema de paquetes.

Como se ha visto anteriormente, un paquete de R es una colección estructurada de código (R, C u otro), documentación, y/o datos para realizar tipos de análisis concretos. Según su contenido, los paquetes se pueden clasificar en:

- Paquetes de código (*software packages*), que proporcionan implementaciones especializadas de métodos estadísticos y gráficos.

- Paquetes de datos de diversos tipos como:
  - Metadatos biológicos (*annotation packages*), que contienen asignaciones (*mapping*) entre diferentes identificadores de genes (por ejemplo, "AffyID", "GO", "LocusID", "PMID"), "CDF" e información de la sonda de secuencia de arrays de Affy. Ejemplos: `hgu95av2.db`, `GO.db`, `KEGG.db`.

  - Datos experimentales (*experiment packages*), que contienen código, datos y documentación para experimentos o proyectos específicos. Ejemplo: `yeastCC` (Spellman y otros, 1988, yeast cell cycle), `golubEsets` (Golub y otros, 2000, ALL/AML data).

- Paquetes de cursos, que contienen código, datos, documentación y ejercicios para el aprendizaje de un curso concreto. Ejemplos: `EMBO03`.

La figura 16 muestra algunos de los paquetes de Bioconductor agrupados por funcionalidades.

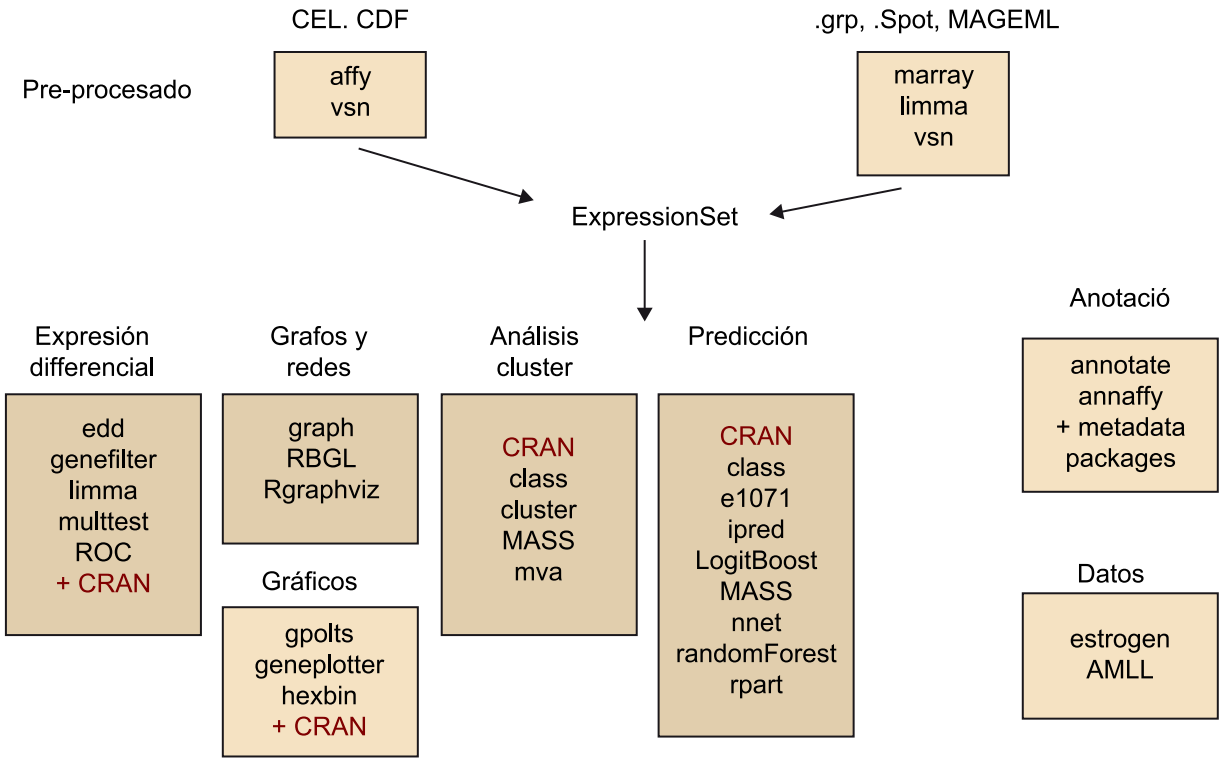

Figura 16. Algunos paquetes de Bioconductor

2.3.3. Programación orientada a objetos en R y Bioconductor

El proyecto Bioconductor adoptó el paradigma de la programación orientada a objetos (OOP) propuesto en 1998 por J.M. Chambers ([3]). Este diseño de clases y métodos orientados a objeto permite una representación eficiente y la manipulación de grandes y complejos conjuntos de datos biológicos de varios tipos. En el paquete methods de R se proporcionan herramientas para la programación mediante este mecanismo de clase/método.

Clases y métodos

Una clase puede definirse como una representación (informática) abstracta de un tipo de objetos que pueden existir del mundo real. La clase refleja cómo imaginamos determinado objeto y qué información debe contener el mismo.

En Bioconductor una clase está formada por:

- Unas componentes (*slots*) reservados para contener determinados datos.
- Unas funciones (*methods*) para manipularlos.

Las clases son representaciones abstractas que se concretan en *objetos*. Un objeto es una variable creada a partir de una clase, por lo que se le suele denominar instancia de la clase. La clase es la que determina la estructura y los valores iniciales que poseerán los objetos.

Un método es una función que realiza una acción sobre los datos (objetos). Cada método define cómo debe comportarse la acción particular dependiendo del tipo de argumentos que tenga. Los métodos permiten cálculos adaptables a cada tipo de datos en particular (es decir, a las clases). Hay funciones genéricas que actúan como un distribuidor (*dispatcher*), examinando los argumentos y determinando qué métodos son los más adecuados para ser invocados en cada caso.

Ejemplos de funciones genéricas en R son `plot`, `print`, `summary`.

El paquete de R `methods` contiene funciones tanto para la definición de nuevas clases y métodos (por ejemplo `setClass` y `setMethod`) como para el trabajo con estos.

Entornos y clausuras

Un entorno (en inglés, *environment*) es un objeto que contiene enlaces entre símbolos y valores. Es muy similar a una tabla *hash*. Los entornos son accesibles a través de las siguientes funciones:

- `ls(env=e)`, para obtener la lista de objetos del entorno e.
- `get('x', env=e)`, para obtener el valor del objeto x en el entorno e.
- `assign('x', y, env=e)`, para asignar a x el valor y dentro del entorno e.

Los entornos pueden asociarse a funciones. Cuando un entorno está asociado con una función, se utiliza para obtener valores de cualquiera de sus variables no vinculadas (*unbound variables*).

El término *clausura* (en inglés, *closures*) se refiere a la unión (vinculación) de la función con el entorno que la contiene. Paquetes de anotaciones, como `annotate`, `genefilter` y otros se basan en el uso de entornos y clausuras.

2.3.4. Dos clases importantes: `expressionSet` y `affyBatch`

Aunque la programación orientada a objetos no se ha impuesto completamente entre los usuarios de Bioconductor, hay algunos conceptos que sí que se han generalizado y que es necesario –o como mínimo muy práctico– conocer para trabajar con Bioconductor. Se trata de la clase `expressionSet` y la clase `affyBatch`, directamente relacionada con ella.

La clase `expressionSet` se usa para representar datos procesados de arrays de dos colores o de un color. A su vez, la clase está compuesta por objetos de otras clases como `assayData` (la matriz de expresiones, de *n* genes por *m* muestras), `phenoData` (las covariables de las muestras, una instancia de la clase `anno-`

tatedDataFrame), annotation (nombre del paquete de anotaciones), description (información en formato MIAME) y notes (para comentarios adicionales).

La figura 17 muestra la estructura interna de la clase expressionSet.

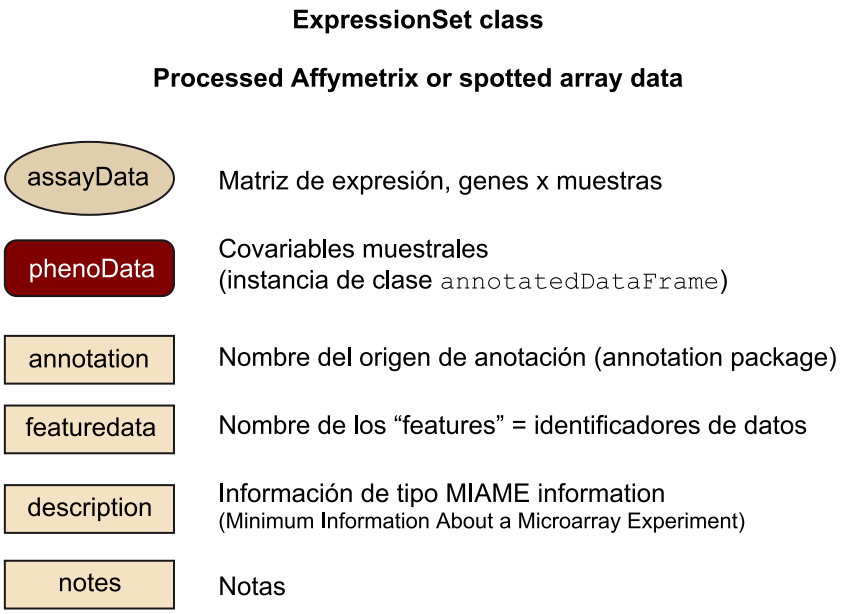

Figura 17. La clase expressionSet

La clase affyBatch se usa para datos a nivel de intensidad de sonda para conjuntos de arrays (mismo CDF), y está integrada por objetos como cdfName (nombre del fichero CDF), nrow y ncol (dimensiones del array), exprs y se.exprs (matrices con las intensidades a nivel de sonda y sus errores estándar), phenoData, annotation, description y notes.

2.3.5. Primeros pasos: instalación, cursos, *vignettes*

Instalación básica

No describiremos aquí la instalación de Bioconductor. Tan solo recordad que se basa en dos instrucciones:

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

En general, los paquetes de R se pueden instalar con la función install.packages. Estos paquetes solo necesitan ser instalados una vez, pero deben ser cargados en cada sesión nueva de R que se abra. Para cargar estos paquetes ya instalados, se usa el comando library de R.

```
> library(Biobase)
```

Para instalar cualquier paquete de Bioconductor, iniciar una sesión de R e introducir:

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

Para citar cualquier paquete desde R, teclear:

```
> citation("nombre_del_paquete")
```

Para actualizar los paquetes existentes, usar la función `update.packages`.

Cursos cortos

Bioconductor ofrece cursos rápidos en forma de elementos modulares de formación sobre software y metodología estadística, notas de conferencias, ejercicios prácticos de computación y otros paquetes de autoformación disponibles en línea.

Las viñetas (*vignettes*)

La viñeta o *vignette* es el sistema de documentación adoptado por Bioconductor que se basa en la idea de documentación reproducible.

Consiste en un documento ejecutable que incluye una colección de bloques de código y bloques de comentarios de texto. Las viñetas ofrecen documentación estadística dinámica, integrada y reproducible, que puede ser actualizada automáticamente si los datos o el análisis cambian.

Se generan usando la función `Sweave` de R.

Cada paquete de Bioconductor contiene al menos una viñeta, que aporta descripciones de sus funcionalidades. Las viñetas se encuentran en el subdirectorio `doc` de cada paquete instalado, y son accesibles desde el menú de ayuda. Pueden ser usadas de forma interactiva y están disponibles también de forma separada en el sitio web de Bioconductor.

Algunas funcionalidades relacionadas con las viñetas son `openvignette` de Biobase (un menú de viñetas disponibles con su visualización en PDF) `vExplorer` de `tkWidgets` (para uso interactivo de las viñetas), y `reposTools`.

Tanto los sitios web de R como de Bioconductor disponen de toda la documentación actualizada así como de listas de correo para resolver dudas de los usuarios.

2.3.6. Listado de paquetes usados o citados en el texto

A continuación os ofrecemos el listado de paquetes usados o citados en el texto:

- **affy (*software package*)**. Métodos para arrays de oligonucleótidos de Affymetrix. El paquete contiene las funciones para el análisis exploratorio de arrays de oligonucleótidos. La dependencia de `tkWidgets` solo se refiere a algunas funciones de conveniencia. El paquete `affy` es completamente funcional sin él.

- **affyPLM (*software package*)**. Métodos para el ajuste de modelos a nivel de sonda ("PLM"). Un paquete que amplía y mejora la funcionalidad del paquete base `affy`. Tiene rutinas que hacen un uso intensivo de código compilado para más velocidad. Se centra en la aplicación de métodos para el ajuste de modelos a nivel de sonda y herramientas que usan estos modelos. Contiene herramientas de evaluación de la calidad basadas en PLM.

- **affyQCReport (*software package*)**. Este paquete crea informes de control de calidad (QC) para objetos `affyBatch`. El informe tiene por objeto permitir al usuario una rápida evaluación de calidad de un conjunto de matrices en un objeto `AffyBatch`.

- **annotate (*software package*)**. Anotación para microarrays. Uso de entornos R para gestionar las anotaciones.

- **arrayQualityMetrics (*software package*)**. Este paquete genera informes de medidas de calidad para contenedores de datos de microarrays de Bioconductor (`ExpressionSet`, `NChannelSet`, `AffyBatch`). El informe contiene tanto secciones generales como específicas de cada plataforma. Es compatible con plataformas de arrays de uno y dos colores.

- **Biobase (*software package*)**. Funciones básicas de Bioconductor que son requeridas por muchos otros paquetes o que reemplazan ciertas funciones de R.

- **CMA (*software package*)**. Este paquete aporta una completa colección de algoritmos basados en microarrays para aprendizaje automático (*machine learning*) y estadística. Selección de variables, ajuste hiperparamétrico, evaluación y comparación, pueden ser ejecutados de forma combinada o paso a paso en un entorno fácil de usar.

- **cMAP (*annotation package*)**. Fichero de anotaciones para `cMAP` reunidas a partir de repositorios de datos públicos.

- **e1071 (*CRAN package*)**. Funciones para el análisis de categorias latentes, transformada de Fourier de tiempo reducido, agrupamiento difuso (*fuzzy*

clustering), máquinas de vectores de soporte (*support vector machines*), cálculo de la ruta más corta, agrupación en bolsas (*bagged clustering*), clasificador 'naive' de Bayes.

- **gcrma (*software package*)**. Ajuste de la hibridación no específica (*background correction*) usando información de secuencia.

- **genefilter (*software package*)**. Algunas funciones básicas para el filtrado de genes.

- **GO.db (*annotation package*)**. Conjunto de mapas de anotación que describen la "Gene Ontology" al completo usando datos de *GO*.

- **golubEsets (*experiment package*)**. Representación de datos de leucemia de Golub con algunos datos de covarianza de procedencia desconocida hasta hoy por el responsable del paquete.

- **gplots (*CRAN package*)**. Herramientas varias de programación R para la representación gráfica de datos.

- **hgu95av2.db (*annotation package*)**. Datos de anotación de "Affymetrix Human Genome U95" (chip hgu95av2) procedentes de repositorios públicos.

- **KEGG.db (*annotation package*)**. Conjunto de mapas de anotación para KEGG procedentes de la misma "Kyoto Encyclopedia of Genes and Genomes".

- **limma (*software package*)**. Análisis de datos, modelos lineales y expresión diferencial para datos de microarrays.

- **multtest (*software package*)**. Procedimientos de pruebas de hipótesis múltiples basados en *Bootstrap* paramétrico y remuestreo por permutación (incluyendo métodos bayesianos empíricos) para el control de las tasas de error como "Familywise error rate (FWER)", "generalized family-wise error rate (gFWER)", "tail probability of the proportion of false positives (TPPFP)" y "false discovery rate (FDR)".

- **plier (*software package*)**. Implementa el algoritmo PLIER de Affymetrix. El método PLIER (*probe logarithmic error intensity estimate*) produce una señal mejorada teniendo en cuenta los patrones experimentales observados en el comportamiento de las sondas y manejando de forma apropiada el error a nivel de los valores de señal extremos.

- **rpart (*CRAN package*)**. Particionamiento recursivo.

- **simpleaffy (*software package*)**. Proporciona funciones de alto nivel para la lectura de archivos .CEL y datos fenotípicos de Affy, haciendo cálculos simples, como "t-tests" y "fold changes". Hace un uso intensivo de la librería `affy`. También tiene algunas funciones de diagramas de dispersión y mecanismos para la generación de figuras de alta resolución para publicaciones.

- **tkWidgets (*software package*)**. `Tk widgets` basados en R. `Widgets` para proporcionar interfaces de usuario. Se requiere la instalación de `tcltk` para que funcionen.

- **yeastCC (*experiment package*)**. `ExpressionSet` de los datos de experimentos del ciclo celular de la levadura, de Spellman y otros (1998).

- **ygs98.db (*annotation package*)**. Datos de anotaciones de "Affymetrix Yeast Genome S98 Array" (chip ygs98) obtenidos a partir de datos de repositorios públicos.

- **ygs98cdf (*annotation package*)**. Paquete con el entorno para la representación del fichero de `YG_S98.cdf` de "Affymetrix Yeast Genome S98".

- **ygs98probe (*annotation package*)**. Datos de secuencias de sondas para los microarrays de tipo ygs98.

3. Fundamentos de estadística

3.1. Introducción

Cuando se realiza un estudio de investigación, el objetivo consiste en poder generalizar los resultados obtenidos en una muestra a una población (proceso conocido como inferencia). Para ello se estudia un reducido número de individuos (o más generalmente unidades experimentales), obtenidos de la población de forma independiente, con la idea de poder generalizar su comportamiento a la población de la cual esa muestra procede. Este proceso de inferencia se efectúa por medio de métodos estadísticos basados en la probabilidad. La población representa un conjunto (grande) de individuos que deseamos estudiar y generalmente suele ser inaccesible. Es, en definitiva, un colectivo homogéneo que reúne unas características determinadas. La muestra es el conjunto menor de individuos (subconjunto de la población accesible y limitado sobre el que realizamos las mediciones o el experimento con la idea de obtener conclusiones generalizables a la población). El individuo es cada uno de los componentes de la población y la muestra. La muestra debe ser representativa de la población, lo que implica que cualquier individuo de la población en estudio debe haber tenido la misma probabilidad de ser elegido.

Las razones para estudiar muestras en lugar de poblaciones son diversas:

- Estudiar la totalidad de los pacientes o personas con una característica determinada en muchas ocasiones puede ser una tarea inaccesible o imposible de realizar.

- Aumentar la calidad del estudio. Al disponer de más tiempo y recursos, las observaciones y mediciones realizadas a un reducido número de individuos pueden ser más exactas y plurales que si las tuviésemos que realizar a una población.

- La selección de muestras específicas nos permitirá reducir la heterogeneidad de una población al indicar los criterios de inclusión y/o exclusión.

Lo que medimos en cada individuo de la muestra son las variables (edad, sexo, peso, talla, tensión arterial sistólica, etcétera). Los datos son los valores que toma la variable en cada caso. Deberemos además concretar la escala de medida que aplicaremos a cada variable.

La naturaleza de las observaciones será de gran importancia a la hora de elegir el método estadístico más apropiado para abordar su análisis. Con este fin, clasificaremos las variables, a grandes rasgos, en dos tipos: cuantitativas y cualitativas.

1) Variables cuantitativas. Estas pueden ser de dos tipos:

a) Variables cuantitativas continuas, si admiten tomar cualquier valor dentro de un rango numérico determinado (edad, peso, talla).

b) Variables cuantitativas discretas, si no admiten todos los valores intermedios en un rango. Suelen tomar solamente valores enteros (número de hijos, número de partos, número de hermanos, etc.).

2) Variables cualitativas. Este tipo de variables, también llamadas categóricas, representan una cualidad o atributo que clasifica a cada caso en una de varias categorías. La situación más sencilla es aquella en la que se clasifica cada caso en uno de dos grupos (hombre/mujer, enfermo/sano, fumador/no fumador). A este tipo de datos se les llama dicotómicos o binarios.

Como resulta obvio, en muchas ocasiones este tipo de clasificación no es suficiente y se requiere un mayor número de categorías (color de los ojos, grupo sanguíneo, profesión). En el proceso de medición de estas variables, se pueden utilizar dos escalas:

a) Escalas nominales. Esta es una forma de observar o medir en la que los datos se ajustan por categorías que no mantienen una relación de orden entre sí (color de los ojos, sexo, profesión, presencia o ausencia de un factor de riesgo o enfermedad).

b) Escalas ordinales. En las escalas utilizadas existe un cierto orden o jerarquía entre las categorías (grados de disnea, estadiaje de un tumor).

3.2. Análisis descriptivo

Una vez que se han recogido los valores que toman las variables de nuestro estudio (datos), procederemos al análisis descriptivo de los mismos.

3.2.1. Variables categóricas

Para variables categóricas, como el sexo, se cuenta el número de casos en cada una de las categorías (frecuencia absoluta), reflejando habitualmente el porcentaje que representan del total (frecuencia relativa), y se expresan en una tabla de frecuencias. Gráficamente se pueden representar mediante los diagramas de barra (*barplots*) y los gráficos de pastel (*piecharts*).

El ejemplo que usamos a continuación muestra cómo leer la secuencia de "X94991.1" de la especie *Homo sapiens* en el GenBank, para la construcción de una tabla de frecuencias y un gráfico de pastel. Para ello, leemos la secuencia a partir del paquete APE de R, que nos proporciona funciones para leer y manipular los árboles filogenéticos y las secuencias de ADN. El número de veces que aparece cada nucleótido se puede mostrar en una tabla de frecuencias. Por ejemplo, para el gen *Zyxin*.

```
> if(!(require("ape")))
+ install.packages(c("ape"),repo="http://cran.r-project.org",dep=TRUE)
> require(ape)
> print(t<- table(read.GenBank(c("X94991.1"),as.character=TRUE)) )


  a   c   g   t
410 789 573 394


> pie(as.numeric(t))
```

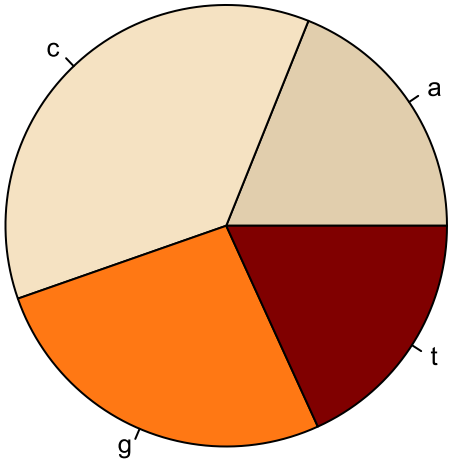

Figura 18. Pie chart o gráfico de pastel
El pie chart o gráfico de pastel es un gráfico de sectores que nos representa el porcentaje en que se dan los valores de una variable.

3.2.2. Variables numéricas

Para variables numéricas en las que puede haber un gran número de valores observados distintos, se ha de optar por un método de análisis diferente, respondiendo a las siguientes preguntas:

1) ¿Cómo se distribuyen los datos?

2) ¿Alrededor de qué valor se agrupan los datos?

3) Si suponemos que se agrupan alrededor de un número, ¿cómo lo hacen? ¿Muy concentrados? ¿Muy dispersos?

3.2.3. Gráficos

Para desarrollar los ejemplos de este subapartado usaremos los datos de expresión génica recogidos por Golub y otros [6]. Una serie de datos del conjunto Golub está contenido en el paquete multtest, que es parte de Bioconductor.

Los datos de este dataset consisten en valores de expresión génica (3.051 filas de genes) de 38 pacientes enfermos de leucemia. A 27 pacientes se les diagnosticó leucemia linfoblástica aguda (ALL) y a 11, la leucemia mieloide aguda (AML). La clase de tumor viene dada por el vector numérico golub.cl, donde ALL se indica con el valor 0 y AML por el valor 1. Los nombres de los genes se recogen en la matriz golub.gnames, donde sus columnas corresponden al número índice de los genes, el ID y el nombre, respectivamente. Estos datos han sido procesados por los procedimientos descritos en Yang y otros (2002) [12].

Histogramas

Una primera aproximación de las variables se puede obtener a partir de la visualización de los datos. Para visualizar los datos con un histograma deberemos dividir el rango de valores en un número de intervalos iguales y representar las frecuencias por un intervalo que se dibujará como un rectángulo de altura proporcional a esta frecuencia. Como ejemplo vamos a crear un histograma a partir de los valores de expresión génica del gen "CCND3 Cyclin D3" de los pacientes con leucemia linfoblástica aguda, es decir aquellos que se etiquetan como "ALL", citados en el estudio de Golub y otros ([6]). Para ello hemos de cargar los datos que se encuentran como ejemplo en la librería multtest mediante la instrucción data(golub). Esta instrucción cargará en memoria la matriz de expresiones golub, junto a otros objetos, entre los que se encuentra golub.cl. Lo primero que se debe realizar es convertir este vector numérico a factor, para luego seleccionar el gen a estudiar (en nuestro caso corresponde a la fila 1.042 de la matriz de expresiones) y poder realizar el histograma.

```
> if(!(require("multtest")))
+      {source("http://bioconductor.org/biocLite.R")
+ biocLite("multtest")
+       }
> require(multtest)
> data(golub)
> gol.fac<-factor(golub.cl,levels=0:1, labels=c("ALL","AML"))



> hist(golub[1042, gol.fac=="ALL"])
```

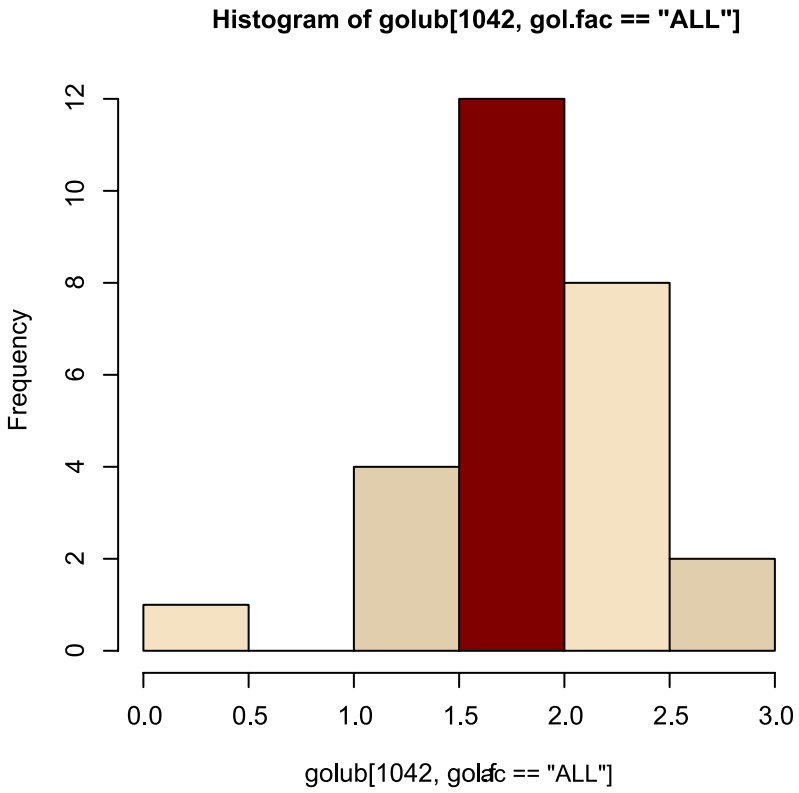

Figura 19. Histograma
Un histograma es la representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados.

En el gráfico anterior podemos observar cómo los valores están más o menos simétricamente distribuidos alrededor de la media.

Boxplot

El diagrama de caja se obtiene ordenando los n valores y calculando los siguientes cuartiles:

- **Cuartil 25% (primer cuartil)**: valor del conjunto de datos para el que el 25% de los datos son menores que él.

- **Cuartil 75% (tercer cuartil)**: valor del conjunto de datos para el que el 75% de los datos son menores que él.

Este gráfico consiste en un rectángulo (caja) que comprende los 2 cuartiles anteriores de cuyos lados (superior e inferior) se derivan dos segmentos respectivamente: uno hacia arriba y uno hacia abajo (llamados bigotes). Este rectángulo está dividido en 2 a partir de la mediana (ver apartado 3.2.4.) de los datos. Más allá de los bigotes o brazos de la caja se hallan las observaciones atípicas o extremas defindas como ls valores que superan 1.5 veces el rango intercuartil.

Para visualizar este gráfico vamos a escoger un ejemplo bioinformático. Con la construcción de dos diagramas de caja para la expresión del gen `CCND3 Cyclin D3`, podemos tener una idea de la distribución de `ALL` y `AML`.

```
> boxplot(golub[1042,] ~ gol.fac)
```

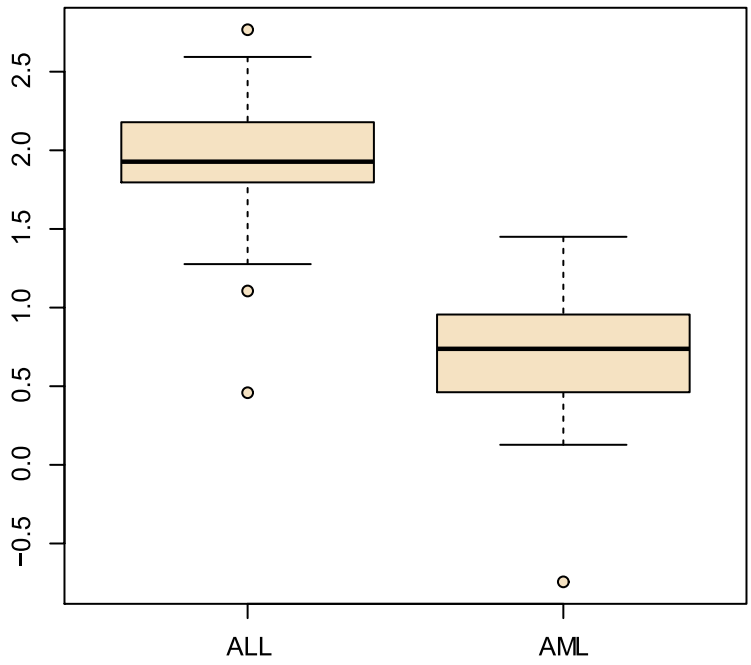

Figura 20. Boxplot
Un boxplot o diagrama de caja es un gráfico en forma de caja que se crea a partir de los cuartiles y que nos permite detectar valores extremos y ver la simetría de una distribución.

Como podemos observar por la posición de las cajas, los valores de expresión génica para `ALL` son mayores que los de `AML`. Además, dado que los dos recuadros alrededor de la mediana son más o menos igual de anchos, podemos decir que los datos están distribuidos razonablemente de forma simétrica alrededor de la mediana.

3.2.4. Estadísticos descriptivos

Existen varias formas para describir la tendencia central, así como la dispersión de los datos. En particular, la tendencia central puede ser descrita por la media y la mediana y la dispersión por la varianza, la desviación estándar y el rango intercuartílico o la desviación absoluta media.

Medidas de tendencia central

Los estadísticos descriptivos de tendencia central más importantes son la media y la mediana. La media muestral de un conjunto de valores: $x_1 \cdots x_n$ se define como:

$$\overline{x} = \frac{1}{n} \sum x_i = \frac{1}{n}(x_1 + \text{- - - -} + x_n)$$

La mediana se define como el segundo cuartil y se denota $X_{0.5}$. Cuando los datos se distribuyen simétricamente alrededor de la media entonces la media y la mediana son iguales. Como los valores extremos no influyen en el tamaño de

la mediana, es muy robusta frente a datos atípicos. La robustez es importante en bioinformática porque los datos están frecuentemente contaminados por valores extremos (*outliers*).

El término *robustez* se utiliza en estadística para hacer referencia a ciertas características deseables de los procesos estadísticos. Se dice que un proceso es robusto respecto de las desviaciones de los supuestos del modelo cuando el proceso continúa funcionando bien, aun cuando, en mayor o menor extensión, los supuestos no se mantienen. En el ejemplo siguiente veremos cómo se calculan los cuartiles de una muestra. Para calcular los valores exactos de los cuartiles podemos usar el código siguiente con una secuencia de 0.00 a 1.00 con intervalos equivalentes a 0.25, como:

```
> pvec<-seq(0,1,0.25)
> quantile(golub[1042, gol.fac=="ALL"],pvec)
```

El primer cuartil es $X_{0.25}$ = 1.796, el segundo $X_{0.5}$ = 1.928 y el tercero $X_{0.75}$ = 2.179.

Medidas de dispersión

Las medidas más importantes de dispersión son la desviación estándar, el rango intercuartílico y la desviación absoluta media. La desviación estándar es la raíz cuadrada de la varianza de la muestra, que se define como:

$$s^2 = \frac{1}{n-1}\sum(x_i - \bar{x})^2 = \frac{1}{n-1}[(x_1 - \bar{x})^2 + \ldots\ldots + (x_n - \bar{x})^2]$$

Por lo tanto, la varianza es el promedio de la diferencia al cuadrado entre los valores de datos y la media de la muestra. La desviación estándar de la muestra es la raíz cuadrada de la varianza y puede interpretarse como la distancia de los datos a la media. La varianza y la desviación estándar no son estadísticos robustos frente a datos atípicos.

El rango intercuartílico se define como la diferencia entre el tercer y el primer cuartil, es decir, $R = x_{0.75} - x_{0.25}$.

3.3. Distribuciones de probabilidad importantes en estadística

Al iniciar el análisis estadístico de una serie de datos, y después de la etapa de detección y corrección de errores, el paso siguiente consiste en describir la distribución de las variables estudiadas y, en particular, de los datos numéricos. Además de las medidas descriptivas correspondientes, el comportamiento de estas variables puede explorarse gráficamente de un modo muy simple. Este

comportamiento se puede estudiar, desde un punto de vista teórico, a partir del conocimiento de las principales distribuciones de probabilidad de las variables.

3.3.1. Distribuciones discretas

La distribución binomial

La distribución binomial es fundamental y tiene muchas aplicaciones en medicina y bioinformática. La distribución binomial se ajusta a ensayos repetidos que generan una variable discreta dicotómica tales como éxito/fracaso, salud/enfermedad, mutación/no mutación, purina/pirimidina, etc.

Cuando se realizan n ensayos, el número de formas de obtener k éxitos viene dado por el coeficiente binomial: $\frac{n!}{k!(n-k)!}$.

La probabilidad binomial de k éxitos de n consiste en el producto de este coeficiente por la probabilidad de k éxitos y la probabilidad de $n-k$ fallos. Sea p la probabilidad de éxito en un único ensayo y x el valor (aleatorio) que denota el número de éxitos. Entonces la probabilidad `P` de obtener k éxitos en n ensayos `(X=k)` se expresará como:

$$P(X=k)=\frac{n!}{k!(n-k)!}p^k(1-p)^{n-k},\ k=0,\ldots,n.$$

Al conjunto de estas probabilidades calculadas para cada posible valor de k se le llama función de densidad de probabilidad. El paquete R tiene la función `dbinom()`, que proporciona el valor de estas probabilidades para cualquier valor de los parámetros n y p, por ejemplo para $x = 0$ y $x = 1$:

```
> n=100
> p=0.01
> dbinom(0,n,p)

[1] 0.3660323

> dbinom(1,n,p)

[1] 0.3697296
```

Podemos hacer un gráfico de la función de densidad probabilidad mediante:

```
> par(mfrow=c(1,1))
```

```
> plot(dbinom(0:100,n,p),xlab="Bits erroneos",type="h")
```

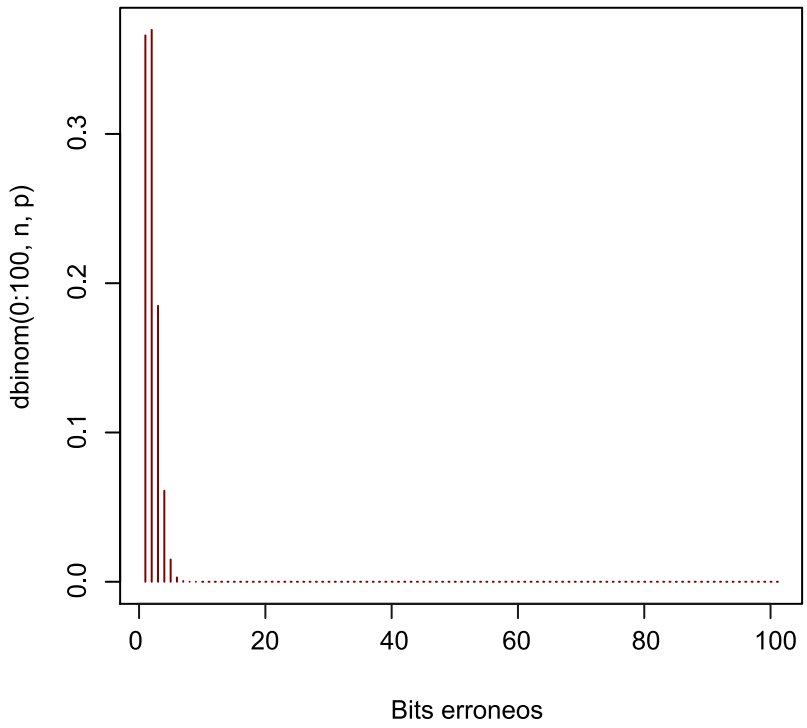

Figura 21. Gráfico de la función de densidad de probabilidad
La función de densidad se calcula a partir de las probabilidades de los valores de la variable.

3.3.2. Distribuciones continuas

Las distribuciones continuas hacen referencia a aquellas variables que pueden tomar cualquier valor en un intervalo. Por lo tanto, el cálculo de la distribución de probabilidades no puede hacerse de la misma forma que en el caso discreto, valor a valor. Es necesario conocer qué función matemática describe el comportamiento probabilístico de la misma. Esta función recibe el nombre de función de densidad y aunque su expresión no es importante en este curso, sí que se dará la forma de la misma para entender el comportamiento de la variable.

La distribución normal

Una de las distribuciones teóricas mejor estudiadas en los textos de bioestadística y más utilizada en la práctica es la distribución normal, también llamada distribución gaussiana. Su importancia se debe fundamentalmente a la frecuencia con la que distintas variables asociadas a fenómenos naturales y cotidianos siguen, aproximadamente, esta distribución. Caracteres morfológicos (como la talla o el peso) o psicológicos (como el cociente intelectual) son ejemplos de variables de las que frecuentemente se asume que siguen una distribución normal. No obstante, y aunque algunos autores han señalado que el comportamiento de muchas variables en el campo de la salud puede ser descrito mediante una distribución normal, en la práctica puede resultar poco frecuente encontrar variables que se ajusten a este tipo de comportamiento.

El uso extendido de la distribución normal en las aplicaciones estadísticas puede explicarse, además, por otras razones. Muchos de los procedimientos estadísticos habitualmente utilizados asumen la normalidad de los datos observados. Aunque muchas de estas técnicas no son demasiado sensibles a desviaciones de la normal y, en general, esta hipótesis puede obviarse cuando se dispone de un número suficiente de datos, resulta recomendable contrastar siempre si se puede asumir o no una distribución normal. La simple exploración visual de los datos puede sugerir la forma de su distribución. No obstante, existen otras medidas, gráficos de normalidad y contrastes de hipótesis, que pueden ayudarnos a decidir de un modo más riguroso si la muestra de la que se dispone procede o no de una distribución normal. Cuando los datos no sean normales, podremos o bien transformarlos o emplear otros métodos estadísticos que no exijan este tipo de restricciones (los llamados métodos no paramétricos).

La distribución normal viene caracterizada por la media y la varianza de la variable, que se acostumbran a expresar con letras griegas: μ (`mu`) para la media y σ^2 (`sigma cuadrado`) para la varianza. La forma de indicar una distribución normal es: $N(\mu;\sigma^2)$.

Estos valores son desconocidos y más adelante se indicará cómo estimarlos. Esta distribución posee ciertas propiedades importantes que conviene destacar:

- Tiene una única moda (valor más probable), que coincide con su media y su mediana.

- La función de densidad (curva normal o gaussiana) es asintótica al eje de abscisas. Por ello, cualquier valor de *y* es teóricamente posible. El área total bajo la curva es, por tanto, igual a 1.

- Es simétrica con respecto a su media. Según esto, para este tipo de variables existe una probabilidad de un 50% de observar un dato mayor que la media, y un 50% de observar un dato menor.

- La distancia entre la línea trazada en la media y el punto de inflexión de la curva es igual a una desviación típica (σ).

- Existe un 95% de posibilidades de observar un valor comprendido en el intervalo $(\mu - 1.96 \cdot \sigma,\ \mu + 1.96 \cdot \sigma)$.

- La forma de la campana de Gauss depende de los parámetros μ y σ.

- La media indica la localización de la campana, de modo que para diferentes valores de μ la gráfica se desplaza a lo largo del eje horizontal. Por otra parte, la desviación estándar determina el grado de apuntamiento de la curva. Cuanto mayor sea el valor de σ, más se dispersarán los datos en torno a la media y la curva será más plana. Un valor pequeño de este parámetro indica, por tanto, una gran probabilidad de obtener datos cercanos al valor medio de la distribución.

La función de densidad viene dada:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e[(x-\mu)^2/2\sigma 2],\ -\infty < x < \infty$$

La forma de calcular probabilidades con una variable continua es a partir de la función de distribución que se define $P(X \leq x)$, es decir, la probabilidad, en la población, de obtener valores menores o iguales a x. Los gráficos de la función de densidad y distribución, son:

```
> plot(seq(-5,5,.001),dnorm(seq(-5,5,.001),0,1),type="l", main="Función de densidad ")
> plot(seq(-5,5,.001),pnorm(seq(-5,5,.001),0,1),type="l", main="Función de Distribución ")
```

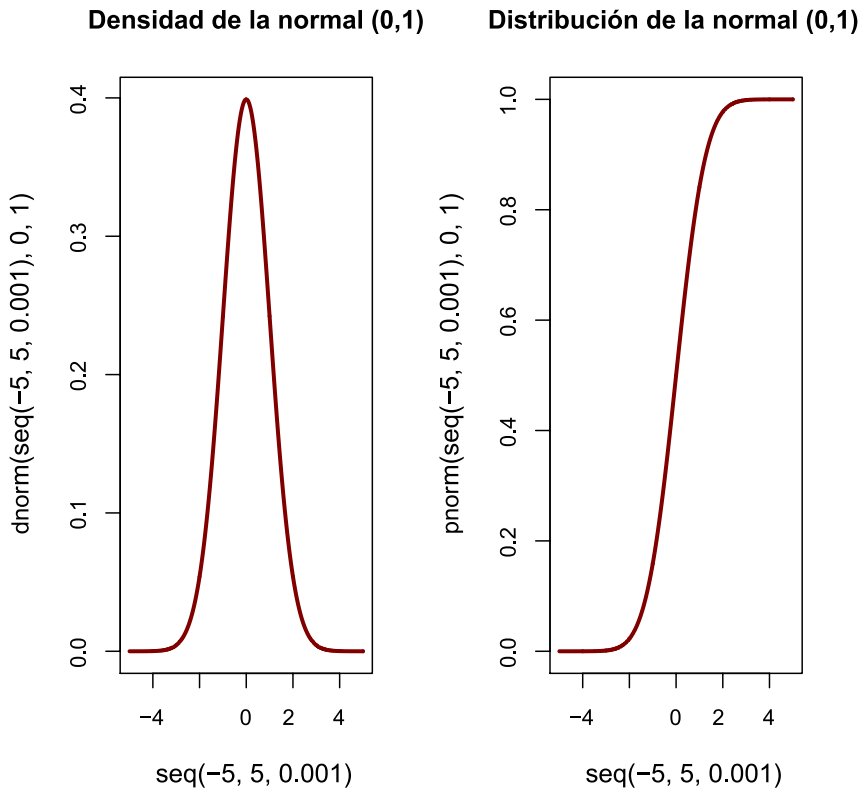

Figura 22. Gráficos de distribución normal(0,1)
La distribución normal, distribución gaussiana o de Gauss es una de las distribuciones continuas que aparece con más frecuencia en los fenómenos reales.

De lo dicho anteriormente se determina que existe una distribución normal para cada valor de la media y la varianza. De entre todas, la que más se utiliza es la `N(0,1)`. Esta variable, que se expresa mediante la letra `Z`, recibe el nombre de normal estandarizada o tipificada y se puede obtener a partir de cualquier normal mediante la transformación: $Z = (X - \mu)/\sigma$.

El Q-Qplot o gráfico de probabilidad normal

El Q-Q (cuantil-cuantil) plot es un método que, entre otros procedimientos, sirve para visualizar la distribución de los valores de una variable. En este gráfico se enfrentan los cuantiles de los valores de la variable en la muestra contra los cuantiles correspondientes a la distribución normal. Se agrega una línea recta que representa los puntos que corresponden exactamente a los cuantiles de la distribución normal. Mediante la observación de la aproximación de los puntos a la línea, se puede evaluar en qué medida los datos se distribuyen normalmente. Es decir, cuanto más cerca de la línea, más probable es que los datos se distribuyan según una normal.

Un ejemplo de cómo producir un Q-Qplot de los valores de expresión génica del gen `CCND3 Cyclin D3` es usando el código siguiente:

```
> qqnorm(golub[1042, gol.fac=="ALL"])
> qqline(golub[1042, gol.fac=="ALL"])
```

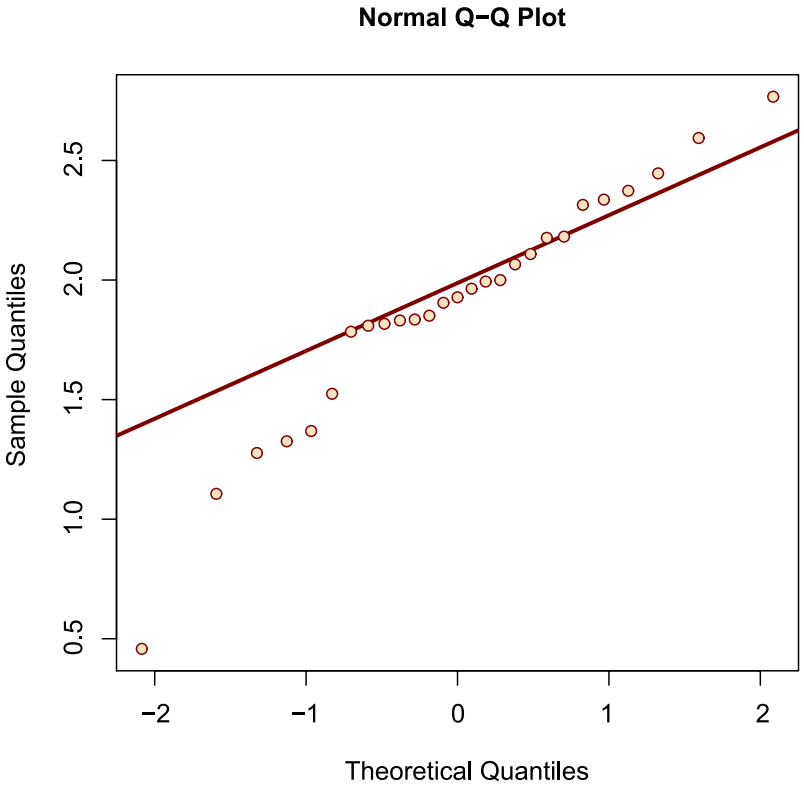

Figura 23. Q-Qplot
El gráfico Q-Qplot es un método gráfico que muestra las diferencias entre la distribución de probabilidad de la que se ha extraído una muestra aleatoria y una distribución normal.

El ejemplo anterior ilustra un caso donde el grado de falta de normalidad es moderado, por lo que no podemos sacar conclusiones.

Distribución Chi-cuadrado

La distribución Chi-cuadrado juega un papel importante en la verificación de hipótesis, tal como veremos en subapartados posteriores. Desde un punto de vista teórico se puede definir de la siguiente manera: Si $Z_1, \ldots, Z_m$ son m variables normales tipificadas e independientes entonces la suma de cuadrados de las mismas:

$$X_m^2 = Z_1^2 + \ldots + Z_m^2 = \sum Z_i^2; i = 1, \ldots . \quad m$$

siguen la distribución Chi-cuadrado con m grados de libertad.

Distribución T

La distribución T (de Student) es una distribución útil en la verificación de hipótesis sobre las medias de las variables, en particular, cuando el tamaño muestral es menor que 30. Si los datos se distribuyen según una normal entonces los valores: $\frac{\sqrt{n}(\overline{x} - \mu)}{s}$ siguen una distribución T con (n - 1) grados de libertad. La distribución T se aproxima a una normal cuando el tamaño de la muestra es 30 o superior.

El gráfico que se genera a continuación es una T de Student que se asemeja a una normal:

```
> ngl<-4
> x<-seq(-4,4,length=5000)
> plot(x,dt(x,ngl), main=c("Densidad t-Student"),sub=paste("gl=",ngl), ylab="t-Student",
+ type="l",col=3)
```

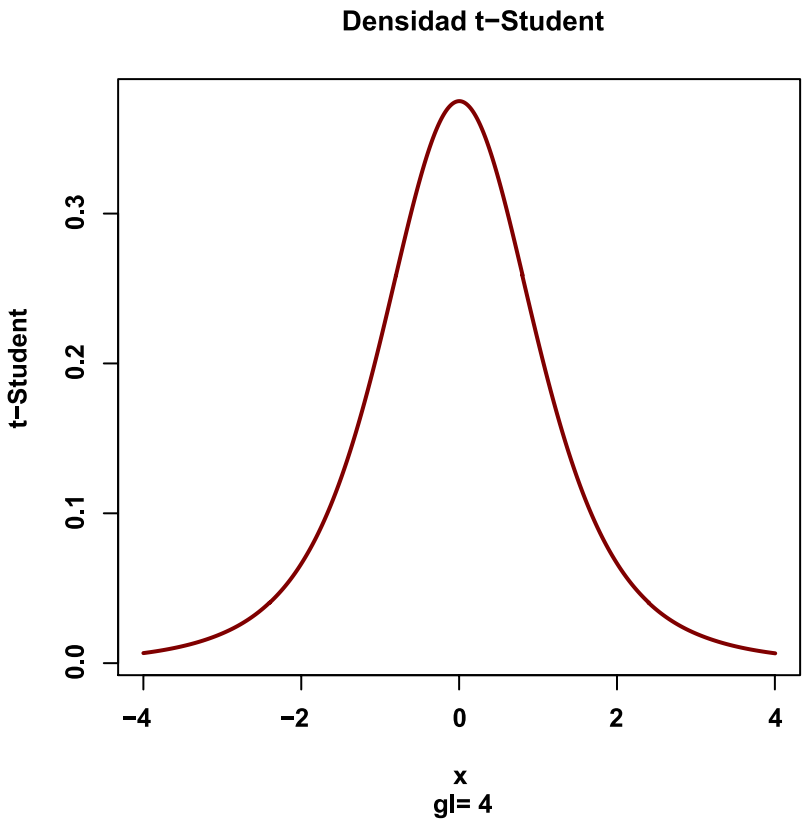

Figura 24. Gráfico T de Student
La T de Student es una distribución relacionada con la inferencia acerca de la media a partir de los datos de una muestra.

Distribución F

La distribución F de Fisher aparece de forma natural cuando se hace inferencia acerca de las varianzas de dos poblaciones. Más específicamente, si las varianzas de dos poblaciones son iguales ($\sigma_1^2 = \sigma_2^2$), se demuestra que el cociente de varianzas muestrales en dos muestras independientes de variables aleatorias distribuidas normalmente, $\frac{s_1^2}{s_2^2}$, sigue una distribución F con (n_1 – 1) y (n_2 - 1) grados de libertad. Donde s_1^2 es la varianza muestral del primer conjunto de n_1 observaciones y s_2^2 la varianza del segundo con n_2 observaciones.

Para finalizar este subapartado en las tablas 3 y 4 haremos un resumen de las funciones de R usadas.

Tabla 3. Resumen de las funciones de R para el cálculo de la densidad

Nombre de la distribución	Parámetros	Densidad
Binomial	(*n*,*p*)	dbinom(*x*,*n*,*p*)
Normal	(μ,σ)	dnorm(*x*,μ,σ)
Chi-cuadrado	*m*	dchisq(*x*,*m*)
T de student	*m*	dt(*x*,*m*)
F de fisher	*m*	df(*x*,*m*,*n*)

Tabla 4. Resumen de las funciones de R para el cálculo la función de distribución, de los cuantiles y la generación de muestras aleatorias de 10 valores de cada distribución

Nombre de la distribución	Distribución	Cuantiles	Muestra aleatoria
Binomial	pbinom(*x*,*n*,*p*)	qbinom(α,*n*,*p*)	rbinom(10,*n*,*p*)
Normal	pnorm(*x*,μ,σ)	qnorm(α,*n*,*p*)	rnorm(10,*n*,*p*)
Chi-cuadrado	pchisq(*x*,*m*)	qchisq(α,*m*)	rchisq(10,*m*)
T de student	pt(*x*,*m*)	qt(α,*m*)	rt(10,*m*)
F de fisher	pf(*x*,*m*,*n*)	qf(α,*m*,*n*)	rf(10,*m*,*n*)

3.4. Inferencia estadística

En los apartados anteriores se utilizaron muchos parámetros poblacionales para definir familias de distribuciones teóricas. En cualquier entorno (empírico) de investigación los valores específicos de estos parámetros son desconocidos, por lo que deben estimarse. Una vez que se dispone de las estimaciones es posible probar estadísticamente hipótesis biológicamente importantes. Existen diferentes métodos para obtener las estimaciones de los parámetros a partir de las muestras. La mayoría se basan en la suposición de que las muestras provienen de poblaciones distribuidas normalmente. De entre los métodos, uno de los más utilizados es el llamado método de la máxima verosimilitud, que se basa en obtener la estimación de los parámetros de forma que se maximiza una función relacionada con la probabilidad de haber obtenido esa muestra concreta. Todos los métodos buscan estimaciones de los parámetros de forma que el estimador cumpla una serie de propiedades que se califican como "buenas", para así garantizar que el valor obtenido se acerca de la mejor forma posible al verdadero valor. La estimación de los parámetros que se han visto hasta ahora, a partir de una muestra son:

- μ: Su estimador es la media muestral $\overline{x}$.
- σ^2: Su estimador es la varianza muestral s^2.

- `p`: Su estimador es la proporción muestral o frecuencia relativa (f/n, siendo `f` el número de veces que se presenta la categoría en la muestra de tamaño `n`).

3.4.1. Contrastes de hipótesis

Un test estadístico es un procedimiento que permite, a partir de una muestra aleatoria representativa, extraer conclusiones que permitan aceptar o rechazar una hipótesis previamente establecida sobre el valor de un parámetro desconocido de una población. La hipótesis establecida se designa por H_0 y se llama hipótesis nula. Esta hipótesis se suele expresar indicando un valor concreto para algún parámetro poblacional ($H_0: \mu = \mu_0$). La hipótesis contraria se designa por H_1 y se llama hipótesis alternativa. Esta hipótesis se basa en la negación de la hipótesis nula y abarca distintas situaciones que dan nombre al tipo de contraste:

- Bilateral: $H_1: \mu \neq \mu_0$
- Unilateral: $H_1: \mu < \mu_0$ o bien $H_1: > 0$

Cada contraste sobre un parámetro lleva asociado un **estadístico de test**, que es una variable aleatoria calculada a partir de la muestra ($T(X_1, \ldots, X_n)$), asociada al test concreto, que describe la distribución de probabilidad del estimador del parámetro. Los pasos a seguir en la resolución de un contraste de hipótesis son los siguientes:

1) Plantear el contraste de hipótesis, definiendo la hipótesis nula y la hipótesis alternativa.

2) Especificar la probabilidad de error que se desea cometer cuando se rechaza la hipótesis nula siendo cierta (se denomina α o nivel de significación). Usualmente se escoge el valor 0.05 como nivel de significación más común.

3) Definir una medida de discrepancia entre la información que proporciona la muestra y la hipótesis H_0. Esta medida de discrepancia se denomina estadístico del contraste y será una función de los datos muestrales y de la información de la hipótesis nula. Este estadístico del contraste debe seguir una distribución conocida bajo H_0, que cumpla:

a) los valores altos sean poco probables cuando H_0 es cierta.
b) los valores pequeños sean altamente probables cuando H_0 es cierta.

4) Decidir qué valores de este estadístico se consideran muy grandes, cuando H_0 es cierta, para que sean atribuibles al azar y por lo tanto se consideran inadmisibles cuando H_0 es correcta. El valor límite que marca la diferencia entre valores grandes y pequeños recibe el nombre de **valor crítico** y su valor queda determinado a partir del nivel de significación.

5) Estimar el parámetro a partir de la muestra y, a partir de él, calcular el valor del estadístico del test ($T_{exp}(X_1, \ldots, X_n)$):

a) Si su valor es pequeño (pertenece a la región de aceptación), entonces se acepta la hipótesis H_0.

b) Si es grande (pertenece a la región de rechazo), entonces se rechaza la hipótesis H_0.

Tipos de error en un contraste de hipótesis

Al realizar un contraste se puede cometer uno de los dos errores siguientes:

- Error tipo I, el que se comete cuando se rechaza la hipótesis nula siendo cierta (*falso positivo*).
- Error tipo II, el que se comete cuando se acepta la hipótesis nula siendo falsa (*falso negativo*).

Debe tenerse en cuenta que solo se puede cometer uno de los dos tipos de error y, en la mayoría de las situaciones, se desea controlar la probabilidad de cometer un error de tipo I. El nivel de significación es precisamente la probabilidad de cometer ese error de tipo I.

Una forma alternativa de tomar la decisión entre ambas hipótesis consiste en calcular el ***p*-valor**. El *p*-valor de un test para contrastar una hipótesis H_0 (aquí, que el gen no está diferencialmente expresado) se define como la probabilidad de obtener un valor del estadístico del test tanto o más extremo que el valor que se ha obtenido sobre la muestra, suponiendo cierta la hipótesis nula, es decir:

$$P\left[T(X_1, \ldots, X_n) \geq T_{exp}(X_1, \ldots, X_n) \middle| H_0\right] = p^*$$

El *p*-valor nos informa sobre cuál sería el nivel de significación α más pequeño que nos hubiera permitido rechazar la hipótesis nula. La decisión consistirá, entonces, en rechazar la hipótesis nula si el *p*-valor es menor o igual al nivel de significación adoptado por el experimentador.

Sin querer profundizar en conceptos teóricos, este enfoque contiene posibles puntos problemáticos que es conveniente conocer para evitar cometer errores por mal uso o abuso de los conceptos. Concretamente:

1) Un *p*-valor bajo conlleva a rechazar H_0, cuando puede ser que sea cierta por azar. Diremos en este caso que hemos declarado un falso positivo.

2) Los *p*-valores no son siempre correctos, dado que su validez depende de que se verifiquen ciertas suposiciones sobre los datos, como la normalidad. Cuando estas suposiciones fallan, los *p*-valores pueden ser completamente incorrectos.

El control de las probabilidades de error

Un test se suele organizar de modo que la probabilidad de obtener falsos positivos esté controlada, es decir, que sea inferior al nivel de significación. Dicho control, sin embargo, no dice nada de la probabilidad de falsos negativos que puede ser muy alta, sobre todo con pequeños tamaños muestrales. Es importante no perder de vista la tabla de decisión (tabla 5) para recordar qué tipos de errores puede conllevar la selección de genes.

Tabla 5

	Decisión		
		Aceptar H_0	**Rechazar H_0**
Realidad	H_0 cierta	Decisión correcta	Error de tipo I
	H_0 falsa	Error de tipo II	Decisión correcta

Las probabilidades de cometer un error son:

$$\underbrace{P(p^* < \alpha | H_0 \text{ cierta})}_{P(\text{Falso positivo (FP)})}, \quad \underbrace{P(p^* > \alpha | H_0 \text{ falsa})}_{P(\text{Falso negativo (FN)})}.$$

Validez de los *p*-valores

Como se ha dicho, los *p*-valores dependen de que se verifiquen ciertas suposiciones como la independencia entre observaciones y normalidad de los datos. En general, lo primero suele ser cierto –o asumible– mientras que lo segundo no tiene porque serlo y además es difícil de verificar en cualquier sentido (con 5 o 10 observaciones, lo que suele ser el caso de muchos estudios de microarrays no se puede hacer una prueba de normalidad de manera fiable.)

En estos casos se puede proceder de dos formas:

- Mirar de comprobar gráficamente, de forma directa o indirecta, la hipótesis de normalidad.

- Recurrir a otros tipos de tests, como tests no paramétricos o tests de permutaciones que no precisan de la suposición de normalidad.

3.4.2. Test t de una muestra

En casi todas las situaciones experimentales la desviación estándar de población, σ, es desconocida. En tales casos, para realizar la prueba $H_0: \mu = \mu_0$ frente $H_1: \mu \neq \mu_0$ nos basaremos en calcular un estadístico de test con una distribución *T*:

$$t_{exp} = \frac{\sqrt{n}\left(\overline{x} - \mu_0\right)}{s}$$

Los pasos a seguir para decidir entre ambas hipótesis son:

- Determinar α.

- Calcular el estadístico del test a partir de la muestra.

- Calcular el *p*-valor asociado a este estadístico.

- Rechazar la hipótesis nula si el *p*-valor es menor que α. La probabilidad de error al tomar esta decisión será α.

3.4.3. Test t de dos muestras con varianzas distintas

Supongamos que se han observado los valores de una variable en dos grupos de pacientes (en dos condiciones experimentales) y que se desea verificar una hipótesis acerca de las medias poblacionales μ_1 y μ_2.

En particular se desea verificar si: $H_0: \mu_1 = \mu_2$ frente a $H_1: \mu_1 \neq \mu_2$.

Este contraste también se puede formular como: $H_0: \mu_1 - \mu_2 = 0$ frente a $H_1: \mu_1 - \mu_2 \neq 0$.

Sean $x_1 \ldots x_n$ y $y_1 \ldots y_m$ las muestras, de de tamaño *n* y *m* respectivamente, de las dos poblaciones. El estadístico t del contraste, con distribución *T* es:

$$t_{exp} = \frac{(\overline{x} - \overline{y}) - (\mu_1 - \mu_2)}{\sqrt{s_1^2/n + s_2^2/m}} = \frac{(\overline{x} - \overline{y})}{\sqrt{s_1^2/n + s_2^2/m}}$$

El criterio de decisión con respecto a la hipótesis nula es idéntico a las pruebas anteriormente mencionadas. Se debe tener en cuenta que el valor de `t` es grande si la diferencia $(\overline{x}\text{-}\overline{y})$ es grande, las desviaciones estándar s_1 y s_2 son pequeñas y los tamaños muestrales son grandes. Esta prueba se conoce como el `test Welch` de dos muestras.

Como ejemplo seguiremos con elos datos obtenidos de Golub y otros ([6]). Se sostiene que el gen `CCND3 Cyclin D3` juega un papel importante con respecto a la discriminación de los pacientes con `ALL` sobre los que tienen `AML`. Intentaremos probar la hipotésis nula de medias iguales y varianzas distintas con el test *t* anterior:

```
> t.test(golub[1042,] ~ gol.fac, var.equal=FALSE)


     Welch Two Sample t-test


data: golub[1042, ] by gol.fac
t = 6.3186, df = 16.118, p-value = 9.871e-06
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
0.8363826 1.6802008
sample estimates:
mean in group ALL mean in group AML
     1.8938826     0.6355909
```

El *p*-valor es 9.87e-06 con lo que rechazaremos la hipótesis de igualdad de medias.

3.4.4. Test t de dos muestras con varianzas iguales

Supongamos que estamos en la misma situación anterior pero ahora se sabe que las varianzas poblacionales (σ_1^2 y σ_2^2) son iguales. Para verificar el mismo test que en el subapartado anterior el estadístico del test se basa en una estimación conjunta de la varianza a partir de las estimaciones en las dos muestras (*pooled sample variance*) que se calcula como:

$$S_p^2 = \frac{(n\text{-}1)s_1^2 + (m\text{-}1)s_2^2}{n+m\text{-}2}$$

El valor del estadístico del test con distribución *T* se calcula en este caso como:

$$t_{exp} = \frac{(\overline{x}\text{-}\overline{y})\text{-}(\mu_1\text{-}\mu_2)}{S_p\sqrt{1/n+1/m}} = = \frac{(\overline{x}\text{-}\overline{y})}{S_p\sqrt{1/n+1/m}}$$

Como ejemplo analizaremos el contraste de hipótesis anterior pero suponiendo varianzas iguales (`var.equal=TRUE`):

```
> t.test(golub[1042,] ~ gol.fac, var.equal = TRUE)

    Two Sample t-test

data: golub[1042, ] by gol.fac
t = 6.7983, df = 36, p-value = 6.046e-08
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
0.8829143 1.6336690
sample estimates:
mean in group ALL mean in group AML
    1.8938826     0.6355909
```

El *p*-valor es de 6.046e-08, la conclusión es rechazar la hipótesis nula de medias poblacionales iguales. En este caso el *p*-valor es ligeramente menor que el de la prueba anterior. En caso de duda sobre la validez de la hipótesis de igualdad de varianzas de la población, se puede comprobar utilizando el siguiente test.

3.4.5. Test F de igualdad de varianzas

El test anterior se basa en que las varianzas poblacionales son iguales. Al ser estas varianzas desconocidas se debe plantear un nuevo test de hipótesis que permita verificar esta suposición. Es decir, se desea verificar: $H_0: \sigma_1^2 = \sigma_2^2$ frente a $H_1: \sigma_1^2 \neq \sigma_2^2$. Para contrastar estas hipótesis se debe partir del cálculo de las varianzas muestrales, y se debe calcular el estadístico f:

$$f_{exp} = \frac{s_1^2}{s_2^2}$$

que sigue una distribución *F* de Fisher con grados de libertad (*n* - 1, *m* - 1).

Si seguimos con el ejemplo anterior, en este caso la hipótesis nula para el gen `CCND3 CyclinD3` será que la varianza de los pacientes con `ALL` es igual a la de los pacientes con ALM y esto se puede probar con la función `var.test()`:

```
> var.test(golub[1042,] ~ gol.fac)

      F test to compare two variances

data: golub[1042, ] by gol.fac
F = 0.7116, num df = 26, denom df = 10, p-value = 0.4652
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
```

```
0.2127735 1.8428387
sample estimates:
ratio of variances
     0.7116441
```

En este caso con un *p*-valor de 0.4652 no podemos rechazar la hipótesis nula de igualdad de varianzas.

3.4.6. Test binomial

Supongamos que estamos interesados en verificar la hipótesis de que la probabilidad de que se presente una característica en la población es un determinado valor p_0, frente al hecho de que esta probabilidad sea mayor. El contraste de hipótesis asociado será: $H_0: p = p_0$ frente a $H_1: p > p_0$. Si el experimento indica que la característica se ha presentado k veces en una muestra de tamaño n y suponiendo una distribución binomial, se podrá verificar la hipótesis nula a partir del cálculo del *p*-valor a partir de la expresión $P(X \geq k)$. Si es menor que el nivel de significación prefijado (0.05 por ejemplo), se rechaza esta.

En el caso de que n sea grande $(n \geq 30)$, se suele recurrir a la propiedad de que la distribución binomial se aproxima a la normal cuando n tiende a infinito.

Por lo tanto el estadístico del test

$$Z = \frac{[(k/n) - p_0]}{\sqrt{p_0(1-p_0)/n}}$$

seguirá una distribución aproximadamente normal. En el caso de que n sea más pequeño que 30 se debe seguir un procedimiento distinto. Supongamos que estamos estudiando secuencias cortas de ARN (microARNs) y nos dicen que se ha obtenido que un microARN de longitud 22-mer contiene 18 purinas. Para probar la hipótesis nula $H_0: p = 0.7$ en contra de $H_1: p > 0.7$ podemos usar la función `binom.test()` de la siguiente forma:

```
> binom.test(18, 22, p = 0.7, alternative = c("greater"), conf.level = 0.95)

      Exact binomial test

data: 18 and 22
number of successes = 18, number of trials = 22, p-value = 0.1645
alternative hypothesis: true probability of success is greater than 0.7
95 percent confidence interval:
0.6309089 1.0000000
sample estimates:
probability of success
```

```
        0.8181818
```

El p-valor de 0.1645 es mayor que el nivel de significación 0.05, por lo que la hipótesis nula no puede ser rechazada.

3.4.7. Test Chi-cuadrado

A menudo se desea verificar una hipótesis sobre más de una probabilidad. Es decir, $H_0:(p_1 \ldots p_m) = (p_{01}, \ldots, p_{0m})$ frente a $H1:(p_1 \ldots p_m) \neq (p_{01}, \ldots, p_{0m})$ donde $p_1 \ldots p_m$ son probabilidades poblacionales correspondientes a m categorías y $p_{01}, \ldots, p_{0m}$ son los valores teóricos que se les suponen. Para cada valor indicado en H_0 se puede calcular el número esperado de ocurrencias en n ensayos independientes: $e_i = np_i$.

El estadístico para verificar el contraste se basa en la diferencia entre los valores observados (o_i) y los esperados (e_i) para cada categoría:

$$q = \sum \frac{(o_i - e_i)^2}{e_i}$$

Este estadístico sigue una distribución Chi-cuadrado con m - 1 grados de libertad. El p-valor se calcula como $P(\chi^2 > q)$.

El test Chi-cuadrado es aplicable si el tamaño muestral es relativamente grande o, lo que es similar, si los valores esperados son mayores que 5. En caso contrario se debe acudir a otros tests como el test exacto de Fisher.

El test exacto de Fisher permite analizar si dos variables dicotómicas están asociadas cuando la muestra a estudiar es demasiado pequeña y no se cumplen las condiciones necesarias para que la aplicación del test chi-cuadrado sea adecuada. Supongamos que tenemos dos categorías dicotómicas, A y B, y la tabla de frecuencias que se muestra en la tabla 6.

Tabla 6. Tabla de frecuencias

	Categoría A	
Categoría B	A1 o Presente	A2 o Ausente
B1 o Presente	f_{11}	f_{12}
B2 o Ausente	f_{21}	f_{22}

Este test se basa en el cálculo del llamado *odds ratio* o razón de probabilidad $f_{11}f_{22}/\left(f_{12}f_{21}\right)$. La hipótesis nula de que las muestras no están asociadas (o bien que la proporción de las categorías de A son similares para cada categoría de B) plantea que el *odds ratio* es igual a 1 y la hipótesis alternativa es que difiere de 1.

Veamos un ejemplo de este test (usado con frecuencia en bioinformática) y del cálculo del *odds ratio*. Vamos a comparar la proporción de oncogenes del cromosoma 1 (B1) con la del genoma completo (B2). En este caso A1 corresponde a la categoría oncogén y A2 a no ser un oncogén. Supongamos que las frecuencias de oncogenes para el cromosoma 1 es igual a $f_{11}=300$ y la de los que no lo son es de $f_{12}=500$. Si consideramos el genoma $f_{21}=3000$ y $f_{22}=6000$, la hipótesis de la que el *odds ratio* es igual a 1 la podemos probar de la siguiente manera:

```
> dat <-matrix(c(300,500,3000,6000),2,byrow=TRUE)
> fisher.test(dat)

     Fisher's Exact Test for Count Data

data: dat
p-value = 0.01912
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
     1.029519      1.396922
sample estimates:
odds ratio
     1.19996
```

Puesto que el p-valor (0.01912) es menor que el nivel de significación, podemos decir que significativamente hay más oncogenes en el cromosoma 1 comparado con el del genoma.

3.4.8. Test de normalidad

Existen diversos procedimientos para probar la hipótesis de que un conjunto de datos se distribuye normalmente. La prueba de Shapiro-Wilks se basa en el grado de linealidad en un gráfico Q-Q.

Como ejemplo del uso de la prueba Shapiro-Wilks probaremos la hipótesis de que el valor de la expresión génica del los genes `ALL` del CCND3 Cyclin D3 se distribuye normalmente:

```
> shapiro.test(golub[1042, gol.fac=="ALL"])

          Shapiro-Wilk normality test
```

```
data: golub[1042, gol.fac == "ALL"]
W = 0.9466, p-value = 0.1774
```

3.4.9. Test de rangos de Wilcoxon

Hemos visto en anteriores subapartados que la comparación de las medias de dos poblaciones normales se realizaba mediante el test *t*. Sin embargo si las poblaciones no se distribuyen siguiendo una distribución normal, debido a una asimetría de la curva de la distribución, el test *t* no es aplicable. En estos casos se utilizan tests basados en los rangos, ya que no requieren suposiciones específicas sobre la distribución de la variable analizada.

Supongamos que deseamos comparar dos poblaciones de las que se han obtenido las muestras: $(x_1, \ldots, x_n)$ con distribución F_x y $(y_1, \ldots, y_n)$ con distribución F_y. Las dos distribuciones F_x y F_y son desconocidas. Si no hay diferencia en el comportamiento de la variable medida en ambas poblaciones (hipótesis nula), las dos distribuciones son idénticas. $H_0 : F_x = F_y$. La idea del test de Wilcoxon-Mann Whitney es la siguiente: si unimos las dos muestras y ponemos los valores en orden, la alternancia entre las X_i y las Y_j debería ser bastante regular. Tendríamos dudas sobre H_0 si los Y_j fuesen en general más grandes que los X_i, o más pequeños, o más frecuentes en ciertos tramos de la sucesión de valores.

El estadístico U de Wilcoxon-Man Whitney se basa en asignar rangos (número de orden en una única muestra obtenida uniendo las dos y ordenando de menor a mayor) a cada valor. Si la hipótesis nula es cierta, la suma de los rangos de las X_i y de las Y_j debería ser parecida. El estadístico de test para esta prueba compara estas sumas para decidir sobre la hipótesis.

Por ejemplo, si queremos realizar la comparación anterior de los grupos ALL y AML pero sin suponer normalidad, el código será:

```
> wilcoxon.test(golub[1042,] ~ gol.fac)
```

3.5. Corrección para pruebas múltiples (*multiple testing*)

Es importante tener en cuenta el efecto que tiene la realización de múltiples tests simultáneos en todos aquellos tests estadísticos que calculan un *p*-valor. Si en un test el *p*-valor es 0.05, entonces se tiene una probabilidad de 5% de dar un error de tipo I (falso positivo), por lo tanto, si realizamos 10000 pruebas a la vez se esperarán unos 500 (=10000×0.05) errores de tipo I (es decir, en 500 tests diremos que hay significación cuando en realidad no es así). Este resultado no puede considerarse aceptable. Una posibilidad es utilizar la llamada **corrección de Bonferroni** para reducir el umbral y detectar significación a un nivel en el que haya solo un 5% de probabilidad de cometer uno o más errores

de tipo I entre todas las pruebas. Este nuevo punto de corte se calcula cómo $0.05/10000 = 5E10\text{-}6$. Este valor es muy estricto, ya que en muchas situaciones se puede admitir un número mayor de errores de tipo I.

Un cambio de filosofía consiste en determinar el número de errores de tipo I (falsos positivos) que se está dispuesto a aceptar al realizar *m* pruebas independientes. El *false discovery rate (FDR*) es un método estadístico utilizado cuando se presentan múltiples hipótesis. En una lista de hipótesis rechazadas el `FDR` controla la proporción esperada de hipótesis nulas incorrectamente rechazadas. El procedimiento para efectuar el control de los falsos positivos mediante el FDR es el siguiente.

Si consideramos $H_1 \ldots H_m$ las hipótesis y $P_1 \ldots P_m$ sus *p*-valores ordenados de manera creciente. Para un FDR (*q*) dado encontraremos el menor *k* a partir del cual:

$$P_k < \frac{k}{m} * q$$

Donde *k* es el número de pruebas aceptadas hasta el momento, *m* es el número total de pruebas realizadas y *q* es la tasa deseada de FDR. Para $k > 1$, esta corrección es menos estricta que realizar la corrección de Bonferroni. Si no se encuentran tests significativos utilizando Bonferroni o la corrección por FDR, entonces se puede mirar a los que tienen el menor *p*-valor. Se puede obtener una estimación de la FDR si se multiplica su *p*-valor por el número de tests múltiples en el experimento.

3.6. Análisis de la varianza

El propósito del análisis de la varianza (ANOVA) es comprobar si existen diferencias significativas entre las medias de varios grupos. Si se comparan solamente dos medias, el ANOVA proporciona el mismo resultado que el test *t*. Sin embargo, a diferencia del test *t*, al comparar k grupos el ANOVA no permite saber cuáles de los grupos son significativamente diferentes entre sí, ya que solo detecta la existencia de estas diferencias. El nombre *ANOVA* se deriva del hecho de que, a fin de verificar una comparación de medias, este análisis se lleva a cabo a partir de la estimación y comparación de unas varianzas específicas. La base del procedimiento consiste en dividir o descomponer la variabilidad total de los datos en componentes de variabilidad debidas a cada uno de los factores que, en nuestro experimento, pueden afectar a la misma. Estas fuentes de variabilidad se cuantifican como sumas de cuadrados de desviaciones respecto a la media.

3.6.1. Análisis de la varianza de un factor

El caso más sencillo es el modelo de un factor, también conocido por *one way anova*, y corresponde a la comparación de *k* grupos de individuos que han recibido tratamientos distintos. El contraste de hipótesis es:

$$\begin{aligned} H_0 &: \mu_1 = \mu_2 = \ldots = \mu_k \\ H_1 &: \mu_i \neq \mu_j \text{ para alguna i, j} \end{aligned}$$

En el modelo ANOVA de un factor se supone que cada observación Y_{ij} (observación *j*-ésima del grupo *i*) puede expresarse como:

$$Y_{ij} = \mu_i + \epsilon_{ij} = \mu + \alpha_i + \epsilon_{ij}$$

A los valores ϵ_{ij} se les llama residuos, y son las desviaciones de cada observación a la media del grupo del que proviene. Se les supone una distribución normal de media 0. Los valores α_i se llaman efectos de cada nivel y representan una medida de la desviación de los datos del grupo *i* respecto a la media global.

La hipótesis nula y alternativa anteriores pueden expresarse como:

$$\begin{aligned} H_0 &: \alpha_1 = \alpha_2 = \ldots = \alpha_k = 0 \\ H_1 &: \alpha_i \neq 0 \text{ para alguna i} \end{aligned}$$

Para contrastar H_0, usaremos la noción de variabilidad, que esencialmente coincide con la de dispersión. Para medir la variabilidad de los datos Y_{ij} utilizaremos las suma de cuadrados totales (*SCT*), y la descompondremos en suma de dos términos: la suma de cuadrados residual o intra-grupos (SCR), que tiene que ver con la variabilidad dentro de cada nivel de factor, y la suma de cuadrados explicada o entre-grupos (SCE), que mide la variabilidad debida a las diferencias entre la media de cada factor y la media global. Más concretamente, se cumple:

$$SCT = SCR + SCE$$

o sea

$$\sum \left(Y_{ij} - \overline{Y}\right)^2 = \sum \left(Y_{ij} - \bar{Y}_i\right)^2 + \sum \left(\bar{Y}_i - \overline{Y}\right)^2$$

siendo $\overline{Y}$ la media de todos los datos y $\bar{Y}_i$ la media del grupo *i*.

Claramente, si H_0 es cierta, entonces SCE será pequeña frente a SCT siendo el porcentaje de variabilidad explicada a $(SCE/SCT) \times 100$.

Cada una de las sumas de cuadrados anteriores permite una estimación de la varianza o *Cuadrado medio*:

- Cuadrado medio entre grupos: $CM_E = \frac{SCE}{k-1}$.
- Cuadrado medio residual: $CM_R = \frac{SCR}{N-k}$, siendo N el número total de datos.

El estadístico de test se basa en la comparación de estas dos estimaciones de la varianza y, por lo tanto, seguirá una distribución F:

$$F = \frac{CM_E}{CM_R}$$

En general podemos decir que H_0 será aceptada si el valor del estadístico se encuentra alrededor del 1. Si es suficientemente mayor que 1, entenderemos que el factor que hemos introducido está realmente explicando las diferencias que observamos entre los valores de la variable Y, y por tanto, que efectivamente hay cierta relación entre Y y el factor que determina los grupos, con lo cual H_0 será falsa. Observemos también que si se rechaza H_0, ello no implica que todas las medias poblacionales sean distintas entre sí, sino simplemente que alguna(s) de ellas es diferente a las demás. De hecho, pueden localizarse los diferentes grupos que aparecen entre los niveles del factor a partir de la realización de comparaciones dos a dos usando un método de comparaciones múltiples.

El ANOVA requiere el cumplimiento de los siguientes supuestos:

- Las distribuciones de probabilidad de la variable Y en cada grupo son normales.
- Las k muestras sobre las que se aplican los tratamientos son independientes.
- Las poblaciones tienen todas igual varianza (homoscedasticidad).

Como ejemplo utilizaremos los datos `ALL` del paquete `ALL` y, específicamente, los valores de expresión del oncogen *1866_g_at* en las células B. Si se realiza un gráfico de los datos se observa que los niveles de expresión difieren en los tres estadios de la enfermedad; por lo tanto, estudiaremos si existen realmente diferencias entre los tres grupos.

```
> if (!(require("ALL")))
+     {source("http://bioconductor.org/biocLite.R")
+       biocLite("ALL")
+     }
> require(ALL)
```

```
> data(ALL)
> ALLB123<-ALL[,ALL$BT %in% c("B1","B2","B3")]
> y<-exprs(ALLB123)["1866_g_at",]
> anova(lm(y~ALLB123$BT))

Analysis of Variance Table

Response: y
          Df Sum Sq Mean Sq F value Pr(>F)
ALLB123$BT 2  5.4563 2.72813 19.848 1.207e-07 ***
Residuals 75 10.3091 0.13745
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> summary(lm(y~ALLB123$BT))

Call:
lm(formula = y ~ ALLB123$BT)

Residuals:
     Min      1Q    Median      3Q     Max
-0.59070 -0.25251 -0.07008 0.16395 1.29635

Coefficients:

              Estimate    Std.  Error t value  Pr(>|t|)
(Intercept)    4.58222 0.08506 53.873  < 2e-16 ***
ALLB123$BTB2  -0.43689 0.10513 -4.156 8.52e-05 ***
ALLB123$BTB3  -0.72193 0.11494 -6.281 2.00e-08 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.3707 on 75 degrees of freedom
Multiple R-squared: 0.3461, Adjusted R-squared: 0.3287
F-statistic: 19.85 on 2 and 75 DF, p-value: 1.207e-07
```

El *p*-valor del test F es 1.207e-07, por lo tanto, se rechaza la hipótesis nula de igualdad de medias. Los coeficientes del modelo representan: `(Intercept)` es el efecto de B1, `ALLB123$BTB2` es la diferencia entre B2 y B1 y `ALLB123$BTB3` es la diferencia entre B3 y B1. De los *t*-tests individuales para estos coeficientes se concluye que la media de B1 es significativamente distinta de 0 y que las diferencias entre B2 y B1 y B3 y B1 también lo son.

3.6.2. ANOVA de más de un factor

En análisis de la varianza se puede extender para permitir al investigador planificar un trabajo y evaluar el efecto combinado de dos o más variables de forma simultánea en el resultado medido, obteniéndose también información

en cuanto a la posible interacción entre los diversos factores. El término *experimento factorial* o *arreglo factorial* se refiere a cómo se asignan los tratamientos que se quieren comparar. Para ello es necesaria una selección previa de los factores a estudiar, sus niveles y la combinación de ellos. La necesidad de estudiar conjuntamente varios factores obedece a la posibilidad de que el efecto de un factor cambie según los niveles de otros factores, esto es, que los factores interactúen o exista interacción.

Si Y_{ijk} es la respuesta para la *k*-ésima unidad experimental del nivel *i* de A y *j* de B, entonces el modelo asociado a un experimento con dos factores: A con *a* niveles y B con *b* niveles, asignados completamente al azar es:

$$Y_{ijk} = \mu + \tau_i + \beta_j + \gamma_{ij} + \epsilon_{ijk}$$

En este modelo τ_i es el efecto del nivel *i* de A, β_j es el efecto del nivel *j* del factor B y γ_{ij} es el efecto de la interacción. Conocer la interacción es más útil que conocer los efectos principales. Una interacción significativa indica que se debe estudiar la diferencia entre los niveles de un factor (por ejemplo de A) en cada nivel del otro factor (de B).

3.7. Introducción a los métodos multivariantes

3.7.1. Análisis de componentes principales

En muchas ocasiones los datos son multidimensionales, es decir, la información de la que se dispone corresponde a la medida de un cierto número de variables (que pueden ser desde una decena a varios miles) sobre los sujetos del estudio. Dado que es imposible discernir tendencias mediante una inspección visual de dicha matriz, es necesario reducir la dimensión para permitir el análisis visual. Como el análisis visual se realiza tradicionalmente en dos dimensiones, en un sistema de coordenadas de *x* e *y*, muchos métodos permiten la reducción de una matriz de cualquier dimensión a solo dos dimensiones.

Si queremos mostrar los datos en solo dos dimensiones, deberemos ser capaces de capturar la mayor cantidad de la variabilidad de los datos que sea posible en tan solo estas nuevas dos dimensiones. El análisis de componentes principales (PCA) ha sido desarrollado con este propósito.

La información que se puede obtener de un análisis de componentes principales depende de si existe una tendencia discernible en los datos que se pueda ver reflejada en dos dimensiones. Como usualmente las matrices de datos tendrán más de 3 variables (único caso en el podríamos representar las observaciones para ver si es factible encontrar las nuevas componentes), será imposible saber *a priori* si existe esta tendencia. Para conocer la "calidad" de la reducción, se calcula el porcentaje de variabilidad retenido por cada compo-

nente. En principio para que la reducción de la dimensión sea efectiva las dos primeras componentes tendrían que retener un porcentaje superior al 50%. Podemos aplicar diferentes criterios a la hora de decidir con cuántas componentes nos quedamos:

1) Uno puede ser a partir del valor de la proporción de variabilidad total de cada componente. Se puede fijar un nivel mínimo y quedarnos con el número de componentes necesario para superar este valor mínimo.

2) Otro criterio puede ser ver en qué momento se produce un descenso de la desviación estándar muy notable. Quedarnos con las componentes previas.

Las nuevas componentes se obtienen como una combinación lineal de las variables originales. En muchas ocasiones es difícil encontrar el significado de estas componentes como variables compuestas, por lo que el uso principal de la técnica es la reducción de la dimensión como paso previo a la aplicación de otros análisis posteriores, por ejemplo, un diagrama de dispersión de las primeras componentes con el objeto de encontrar agrupaciones en los datos o con el objeto de contrastar similitudes o diferencias entre los individuos. Las instrucciones de R para realizar un PCA básico con los datos de golub son:

```
> golub.pca = prcomp(golub, scale = TRUE, center = TRUE)
> summary(golub.pca)
> plot(golub.pca)
```

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. Ambas opciones son necesarias si las variables medidas presentan un rango muy distinto de valores.

3.7.2. Análisis de conglomerados

El objetivo del análisis cluster (*cluster analysis*) es obtener grupos de objetos de forma que, por un lado, los objetos pertenecientes a un mismo grupo sean muy semejantes entre sí, es decir, que el grupo esté cohesionado internamente y, por el otro, los objetos pertenecientes a grupos diferentes tengan un comportamiento distinto con respecto a las variables analizadas, es decir, que cada grupo esté aislado externamente de los demás grupos. Una vez establecidas las variables y los objetos a clasificar, el siguiente paso consiste en establecer una medida de proximidad o de distancia entre ellos que cuantifique el grado de similaridad entre cada par de objetos.

Las medidas de proximidad, similitud o semejanza miden el grado de semejanza entre dos objetos, de forma que, cuanto mayor es su valor, mayor es el grado de similaridad existente entre ellos y con más probabilidad los métodos de clasificación tenderán a ponerlos en el mismo grupo. Cuando las variables son cuantitativas, una de las medidas que se puede utilizar es el coeficiente de correlación de Pearson.

Las medidas de disimilaridad, desemejanza o distancia miden la distancia entre dos objetos, de forma que, cuanto mayor sea su valor, más diferentes son los objetos y menor la probabilidad de que los métodos de clasificación los pongan en el mismo grupo.

Como las medidas de distancia son sensibles a la diferencia de escalas o de magnitudes hechas entre variables, es necesaria la estandarización de datos, cuando estos corresponden a variables cuantitativas, para evitar que las variables con una gran dispersión tengan un mayor efecto en la similaridad. Cuando las variables que se miden son cuantitativas, una de las medidas de distancia más populares es la distancia euclídea. Si se han medido *p* variables en cada objeto, la distancia euclídea se calcula como:

$$d_{ij} = \sum_{k=1}^{p} \left(X_{ik} - X_{jk}\right)^2$$

Métodos jerárquicos

En los métodos jerárquicos, inicialmente cada objeto a agrupar es un grupo en sí mismo y sucesivamente se van fusionando grupos cercanos hasta que todos los individuos confluyen en un solo grupo. Los métodos jerárquicos pueden dividirse en aglomerativos y divisivos. En los primeros se parte de tantas clases como objetos tengamos que clasificar y en pasos sucesivos vamos obteniendo clases de objetos similares; mientras que en los segundos se parte de una única clase formada por todos los objetos, que se va dividiendo en grupos más pequeños sucesivamente. La principal ventaja de los métodos jerárquicos aglomerativos es que se puede representar el problema en forma de árbol o dendrograma, donde se observa muy bien la solución final.

Un algoritmo básico aglomerativo es:

- Calcular la matriz de distancias entre los objetos.
- Identificar cada objeto con un clúster.
- Combinar los dos grupos más cercanos y actualizar la matriz de distancias.
- Repetir el paso anterior hasta que solo quede un grupo.

Como ejemplo usaremos los datos de Golub y otros (1999). Se sospecha que la expresión de los genes "CCND3 CyclinD3" y "Zyxin" difieren entre los grupos `ALL` y `AML`. Vamos a estudiar si, en nuestros datos, las expresiones de estos dos genes permiten confirmar la afirmación anterior.

```
> clusdata<-data.frame(golub[1042,],golub[2124,])
> colnames(clusdata)<-c("CCND3 Cyclin
+ D3","Zyxin")
> plot(hclust(dist(clusdata,method="euclidian"), method="single"))
```

En la figura 25 puede observarse la formación de los clústeres. Si cortamos el dendograma a una altura de 0.5 nos aparecen 5 clústeres, tres de ellos formados por un único elemento.

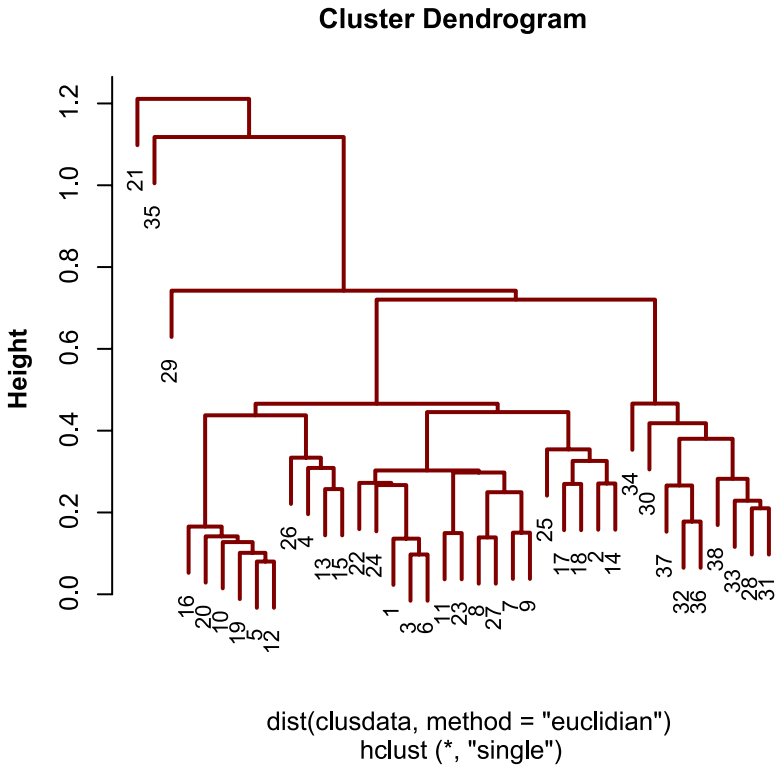

Figura 25. Dendograma
El dendograma es una representación de los clústeres

Método K-means

El método K-means es una alternativa a los métodos jerárquicos, en el cual no es necesario realizar el cálculo inicial de las distancias entre todos los objetos. Existen varias formas de implementarlo pero todas ellas siguen básicamente los siguientes pasos:

1) Se seleccionan *k* centroides o semillas donde *k* es el número de grupos deseado.

2) Se asigna cada observación al grupo cuya semilla es la más cercana.

3) Se calculan los puntos semillas o centroides de cada grupo.

4) Se iteran los 2 pasos anteriores hasta que se satisfaga un criterio de parada, como por ejemplo los puntos semillas apenas cambian o los grupos obtenidos en dos iteraciones consecutivas son los mismos.

El método suele ser muy sensible a la solución inicial dada, por lo que es conveniente utilizar una que sea buena. Una forma de construirla es a partir de una clasificación obtenida por un algoritmo jerárquico.

Bibliografía

[1] **A. Barrier; P. I. Boelle; A. Lemoine; A. Flahault; S. Dudoit; M. Huguier** (2007). "Gene expression profiling in colon cáncer". *Bull Acad Natl Med* (vol. 6, núm. 191, págs.1091-101).

[2] **M. J. Callow; S. Dudoit; E. L. Gong; T. P. Speed; E. M. Rubin** (2000). "Microarray Expression Profiling Identifies Genes with Altered Expression in HDL-Deficient Mice". *Genome Research*.

[3] **J. M. Chambers** (1998). *Programming with data: a guide to the S language*. Springer.

[4] **L. Dyrskjøt; T. Thykjaer; M. Kruhøffer; J. L. Jensen; N. Marcussen; S. Hamilton-Dutoit; H. Wolf; T. F. Ørntoft** (2002). Identifying distinct classes of bladder carcinoma using microarrays. *Nature Genetics* (núm. 33, págs. 90-96).

[5] **Frantz, Simon** (2005). "An array of problems". *Nat. Rev. Drug. Discov*. (vol. 5, núm. 4, págs. 362-363).

[6] **T. R. Golub; D. K. Slonim; P. Tamayo; C. Huard; M. Gaasenbeek; J. P. Mesirov; H. Coller; M. L. Loh; J. R. Downing; M. A. Caligiuri; C. D. Bloomfield; E. S. Lander** (1999). "Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring". *Science* (núm. 286, págs. 531-537).

[7] **I. A. Hedenfalk; M. Ringnér; J. M. Trent; A. Borg** (2002). Gene expression in inherited breast cancer. *Adv Cancer Res* (núm. 84, págs. 1-34).

[8] **H. Ledford** (2008). The death of microarrays? *Nature News* (vol. 7215, núm. 455, págs. 847-847).

[9] **I. Lesur Kupin** (2005). *Study of the Transcriptome of the prematurely aging dna-2 yeast mutant using a new system allowing comparative DNA microarray analysis*. PhD thesis, Universite Bordeaux I.

[10] **S. M. Lin; J. Devakumar; W. A. Kibbe** (2006). Improved prediction of treatment response using microarrays and existing biological knowledge. *Pharmacogenomics* (vol. 3, núm. 7, págs. 495-501). PMID: 16610959.

[11] **M. Schena** (1999). *Microarray Analysis*. J. Wiley & Sons, Hoboken, NJ.

[12] **Y. H. Yang; S. Dudoit; P. Luu; D. M. Lin; V. Peng; J. Ngai; T. P. Speed** (febrero, 2002). "Normalization for cDNA microarray data: a robust composite method addressing single and multiple slide systematic variation". *Nucleic acids research* (vol. 30, núm. 4 e15).

[13] **Zhong Wang; Mark Gerstein; Michael Snyder** (2009). "RNA-Seq: a revolutionary tool for transcriptomics". *Nat Rev Genet* (vol. 1, núm. 10, págs. 57-63).