El paquete `methylearning` y su aplicación *Shiny*: selección de marcadores y clasificación a partir de datos de metilación del DNA

David Piñeyro Valerio

Máster universitario en Bioinformática y bioestadística UOC-UB
Desarrollo de herramientas de soporte a la ómica

Consultor: Antonio Jesús Adsuar Gómez
Profesor responsable: María Jesús Marco Galindo

6 de junio de 2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*El paquete `methylearning` y su aplicación Shiny* asociada: selección de marcadores y clasificación a partir de datos de metilación del DNA
Nombre del autor:	*David Piñeyro Valerio*
Nombre del consultor:	*Antonio Jesús Adsuar Gómez*
Nombre del PRA:	*María Jesús Marco Galindo*
Fecha de entrega (mm/aaaa):	06/2018
Titulación:	*Máster universitario en Bioinformática y bioestadística UOC-UB*
Área del Trabajo Final:	*Desarrollo de herramientas de soporte a la ómica*
Idioma del trabajo:	*Castellano*
Palabras clave:	*Metilación, clasificación, selección de marcadores*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

La aplicación de técnicas de *Machine Learning*, en el contexto del diagnóstico y la prevención, es un campo de gran interés en la actualidad. De entre los datos biológicos que se pueden emplear en este escenario, los datos masivos generados por las técnicas de alto rendimiento actuales representan una gran oportunidad para el avance de la medicina. Entre ellos, los datos epigenéticos han demostrado su utilidad para el diagnóstico de muchas enfermedades de gran impacto social, tales como el cáncer o las enfermedades neurodegenerativas. Sin embargo, la construcción de un modelo de clasificación basado en este tipo de datos es un proceso laborioso, que requiere de un fino equilibrio entre las técnicas de selección de variables, imprescindibles para datos de alta dimensionalidad, y las técnicas de clasificación. Con el fin de proporcionar un marco de trabajo completo, robusto y sencillo para el proceso de creación de modelos de clasificación basados en datos epigenéticos, se ha creado el paquete `methylearning`. Programado en `R`, está pensado para asistir en las tareas de selección de marcadores y clasificación, facilitando la exploración de multitud de combinaciones y generando una gran riqueza de datos, tanto númericos como gráficos, con los que contrastar el rendimiento de diferentes algoritmos. Además, para facilitar el acceso a la herramienta a un público más amplio, también se ha diseñado una aplicación web, basada en *Shiny*, que proporciona toda la funcionalidad del paquete `methylearning`, sin necesidad de tener conocimientos de programación.

Abstract (in English, 250 words or less):

The application of Machine Learning techniques, in the diagnostic and prevention context, is a hot topic nowadays. From all the biological data that can be used for this purpose, those generated by hight throughput techniques represent a great opportunity for the progress in medicine. Among them, epigenetic data has demonstrated its potential for the diagnostic in big social impact diseases, such as cancer and neurodegenerative diseases. However, the process of building a classifier model based on this kind of data is a hard process, which requires a fine balance between feature selection techniques, a necessity due to the dimensionality problem of the data, and the classification techniques. To provide a complete, robust and easy-to-use framework to assist the creation of classifiers based on epigenetic data, we created the `methylearning` package. Programmed in R, it is meant to assist in the process of feature selection and classification, making simple the exploration of multiple combinations and producing a wealth of information, numerical and graphical, to assess the performance of different algorithms. In addition, to open the use of the `methylearning` tool to a wider audience, a Shiny web application was also designed, which incorporates all the `methylearning` functionality, behind a easy-to-use graphical interface.

Índice general

1. Introducción **1**
- 1.1. Contexto y justificación del Trabajo . . . 1
- 1.2. Objetivos del Trabajo . . . 2
- 1.3. Enfoque y método seguido . . . 3
- 1.4. Planificación del Trabajo . . . 4
  - 1.4.1. Tareas y duración . . . 5
  - 1.4.2. Calendario . . . 5
  - 1.4.3. Hitos . . . 6
- 1.5. Breve sumario de productos obtenidos . . . 7
- 1.6. Breve descripción de los otros capítulos de la memoria . . . 8

2. El paquete `methylearning` y su aplicación *Shiny* **9**
- 2.1. Introducción . . . 9
  - 2.1.1. La cromatina, epigenética y salud . . . 9
  - 2.1.2. Técnicas epigenómicas . . . 11
  - 2.1.3. *Machine learning* y epigenética . . . 14
  - 2.1.4. Algoritmos de *Feature Selection* . . . 16
  - 2.1.5. Algoritmos de clasificación . . . 18
  - 2.1.6. Validación del modelo . . . 25
- 2.2. Métodos . . . 26
  - 2.2.1. Diseño del paquete `methylearning` . . . 26
  - 2.2.2. La aplicación *Shiny* de *methylearning* . . . 33
- 2.3. Análisis de demostración: clasificación de pacientes de ALL según su pronóstico . . . 36
  - 2.3.1. Preparación de los datos . . . 37
  - 2.3.2. *Feature selection* . . . 39
  - 2.3.3. Clasificación y validación de los modelos . . . 48
  - 2.3.4. Exploración de los resultados . . . 49
- 2.4. Discusión . . . 63

3. Conclusiones **67**

4. Glosario **69**

5. Bibliografía **71**

Índice de figuras

1.1. Diagrama de Gantt 6

2.1. Multiples niveles de compactación de la cromatina. 10
2.2. *Microarrays* de metilación de Illumina 12
2.3. Representación de un hiperplano. 21
2.4. MMH y vectores de soporte. 21
2.5. Esquema de funcionamiento del paquete *methylearning*. 27
2.6. Mensaje de bienvenida de la aplicación web. 34
2.7. Primera parte del análisis: carga de los datos. 34
2.8. Methylearning Shiny App: FS 35
2.9. Methylearning Shiny App: clasificación 36
2.10. Diagrama de Venn ANOVA vs limma 45
2.11. Diagrama de Venn de 4 FS 46
2.12. Tiempo de computación FS 47
2.13. `Accuracy` en los datos de entrenamiento 58
2.14. `Kappa` en los datos de entrenamiento 59
2.15. ROC boruta + knn 60
2.16. ROC limma + rf 61
2.17. Tiempo de cálculo: clasificación 62
2.18. Tiempo de cálculo: FS + clasificación 63

Índice de cuadros

1.1. Relación de tareas y su duración. 5
1.2. Relación de hitos con sus fechas de entrega. 7

2.1. *Microarrays* vs WGBS 13
2.2. Rendimiento del modelo limma + rf 65

Capítulo 1

Introducción

1.1. Contexto y justificación del Trabajo

El reciente auge de la genómica de alto rendimiento ha puesto a disposición de los investigadores una enorme cantidad de datos con gran potencial para la mejora del diagnóstico y tratamiento personalizado de multitud de patologías. Así, a modo de ejemplo, actualmente se dispone de repositorios gratuitos y de libre acceso tales como el proyecto *The Cancer Genome Atlas* (TCGA), donde se pueden encontrar datos de unos 40 proyectos, englobando 61 tipos de tumores, para un total de 32555 casos (fuente:`https://portal.gdc.cancer.gov/`, *Data Release 10.1 - February 15, 2018*) o el portal *Gene Expression Omnibus* (GEO) del *National Center for Biotechnology Information* (NCBI), donde a fecha de 6 de junio de 2018 se pueden encontrar datos de más de 2,5 millones de muestras provenientes de más de casi cien mil experimentos "ómicos" de todo tipo. Esta enorme cantidad de datos disponibles ha generado una situación nueva en la historia de la ciencia, ya que la capacidad de generar datos ha sobrepasado ampliamente la capacidad de analizarlos. Por ello, las técnicas de *machine learning* (ML, aprendizaje automático) resultan fundamentales hoy en día, consiguiendo generar nuevo conocimiento a partir toda esta cantidad de datos.

En este contexto, actualmente las técnicas de clasificación basadas en datos "ómicos" están revolucionando el diagnóstico de diversas patologías, consiguiendo niveles de precisión y velocidad sin precedentes [1,2]. Por datos "ómicos" se entiende aquellos generados por técnicas de nueva generación, que pretenden abarcar el conjunto completo de un tipo de biomoléculas. Así, existe la genómica, que engloba el conjunto del genoma, la transcriptómica, que abarca el conjunto de RNAs (ribonucleic acids), la epigenómica, para el conjunto de modificaciones químicas que se dan en la cromatina, la proteómica, la metabolómica y nuevas "ómicas" que se van

añadiendo a medida que aparecen nuevas técnicas de caracterización masivas.

Así pues, la aplicación de modelos y métodos del campo del *machine learning* es algo imprescindible hoy en día para muchos proyectos de investigación. Sin embargo, la elección de los algoritmos adecuados y de la metodología concreta para la construcción de estos modelos no es un tema trivial. Por un lado, está el problema de la dimensionalidad de los datos, con un altísimo número de variables para un reducido número de muestras. Esto convierte a la etapa de *feature selection* (FS, selección de marcadores) en esencial para asegurar un buen rendimiento. Por otro lado, existen multitud de algoritmos de clasificación, cuyo rendimiento es muy dependiente del tipo de datos de entrada y del problema que se desea resolver. Adicionalmente, estas dos etapas están estrechamente interrelacionadas, afectando en conjunto al rendimiento del modelo.

El presente trabajo final de máster (TFM) pretende generar una herramienta para facilitar el estudio de la interrelación entre diferentes técnicas de *feature selection* y de clasificiación, ayudando así a explorar multitud de combinaciones, con el fin de encontrar el modelo óptimo para un conjunto de datos determinado. Como se ha comentado, dentro del término datos "ómicos" se engloban datos muy diferentes, obtenidos por multitud de técnicas. Esto hace que el diseño de una herramienta generalista que permita el análisis de todo tipo de datos "ómicos" sea una empresa demasiado ambiciosa para el tiempo disponible. Por lo tanto, el presente TFM se centrará en datos epigenéticos, más concretamente, en datos de metilación del DNA generados a partir de técnicas de *microarrays* de metilación y de secuenciación de alto rendimiento [3,4]. En la actualidad este tipo de datos son muy abundantes, relativamente fáciles de conseguir y de un gran potencial para ser usados en modelos de clasificación automática, especialmente aquellos enfocados al diagnóstico y/o predicción de tratamientos [5].

1.2. Objetivos del Trabajo

1. Implementar un completo *framework*, en forma de paquete del lenguaje de programación `R`, que permita aplicar un conjunto de técnicas de *feature selection* y clasificación, empleando datos de metilación del DNA. Este paquete deberá tener las siguientes características:

    1.1. Aceptar datos de entrada tanto de *microarrays* de metilación de DNA como de secuenciación masiva de bisulfito, en forma de datos preprocesados y normalizados, consistentes en el nivel de metilación por sitio y muestra.

    1.2. Implementar un conjunto de algoritmos de *feature selection*. Estos pueden

ser, tanto algoritmos previamente implementados por otros desarrolladores, como diseñados expresamente para el presente proyecto.

1.3. Implementar un conjunto de algoritmos de clasificación.

1.4. Implementar un sistema de validación de los modelos de clasificación, para evitar el sobreajuste a los datos de entrenamiento.

1.5. Proporcionar los mecanismos para probar las combinaciones deseadas de estos métodos de *feature selection* y clasificación.

1.6. Proporcionar un conjunto de medidas de evaluación del rendimiento, tanto numéricas como gráficas.

2. Desarrollar una aplicación web, usando el paquete *Shiny* (https://shiny.rstudio.com/), que permita hacer uso del paquete desarrollado. Esta herramienta deberá tener las siguientes características:

   2.1. Acceso a las funcionalidades más relevantes del paquete desarrollado.

   2.2. Posibilidad de generar salida tanto numérica como gráfica.

   2.3. Facilitar la posibilidad de descargar los resultados más relevantes, para su posterior uso.

1.3. Enfoque y método seguido

Para la implementación de la herramienta propuesta, se ha escogido el lenguaje de programación `R` [6] por numerosas razones, entre las que destacan:

- Software *open source* con una enorme comunidad de soporte.
- Multitud de paquetes existentes para la implementación de diferentes algoritmos de *machine learning*.
- Marcada orientación estadística y facilidades para la representación gráfica.

Otra alternativa válida, por conocimientos y potencia de las herramientas, hubiera sido el lenguaje de programación Python, empleando librerías tan completas como `scikit-learn` (http://scikit-learn.org/stable/) y otras con tanto potencial como `TensorFlow` (https://www.tensorflow.org/). No obstante, la elección final ha sido `R`, que además proporciona muchas facilidades para la implementación de la interfaz gráfica, gracias al paquete *Shiny* [7].

Del variado conjunto de datos ómicos existentes actualmente, se han escogido los datos de metilación del DNA por las siguientes razones:

- Experiencia previa en el tratamiento de este tipo de datos.
- Facilidad de obtención de una gran cantidad de datos para el desarrollo y puesta a punto de la herramienta.
- Idoneidad para su uso en tareas de clasificación enfocadas al diagnóstico, ya que el perfil epigenómico es muy informativo en este sentido.

Para la elección del conjunto de algoritmos de *feature selection* y clasificación implementados se ha recurrido a los más utilizados en la literatura, para tipos de datos similares. En el caso de los algoritmos de *feature selection*, dada la gran importancia de estos en contextos de alta dimensionalidad, se han implementado multitud de algoritmos, algunos previamente disponibles en otros paquetes de `R` y otros implementados expresamente para el presente proyecto, los cuales se citan a continuación:

- Algoritmo basado en test ANOVA.
- Algoritmo basado en test regresión lineal con limma [8].
- Algoritmo genético.
- Algoritmo de selección secuencial de variables modificado.

En el caso de los algoritmos de clasificación, se ha usado la completa y eficiente implementación que proporciona el paquete `caret` [9], que además facilita enormemente la fase de validación de los modelos.

1.4. Planificación del Trabajo

Como medios necesarios para la realización del proyecto, se ha empleado un ordenador relativamente modesto en cuanto a especificaciones:

- CPU: Intel Core i5 7200U 2.5GHz (2 cores, 4 threads).
- RAM: 8 GiB DDR4.
- Local Disk: 256 GiB SSD.
- OS: Ubuntu 17.10.
- R version: 3.4.4.

No obstante, también se ha tenido acceso puntual a un servidor de alto rendimiento, con 4 x AMD Opteron (4 CPUs, 64 cores, 64 threads) y 512 GiB DDR3 RAM, empleándolo para pruebas con sets de datos de gran tamaño.

1.4.1. Tareas y duración

La planificación inicial de las tareas resultó ser poco acertada en algunos apartados y requirió de modificaciones. Afortunadamente, se replanteó tras el primer informe de seguimiento, logrando mejorar el ritmo de desarrollo y facilitando su conclusión. La planificación final, en cuanto a tareas y duración, es la siguiente:

TAREA	**DURACIÓN**
1. Fase inicial de documentación	30h
1.1. Elección del conjunto de algoritmos de *feature selection* y clasificación a implementar	
2. Desarrollo de la herramienta	120h
2.1. Ajustes de rendimiento y pruebas	
3. Redacción del informe de seguimiento 1 (PEC 2)	15h
4. Implementación de la interfaz gráfica	60h
4.1. Ajustes de rendimiento y pruebas	
4.2. Redacción del informe de seguimiento 2 (PEC 3)	
5. Redacción de la memoria (PEC 4)	50h
6. Preparación de la presentación en diapositivas y video (PEC 5a)	35h
7. Preparación y defensa pública (PEC 5b)	20h
TOTAL	330h

Cuadro 1.1: Relación de tareas y su duración.

1.4.2. Calendario

A continuación se presenta un diagrama de Gantt (figura 1.1) donde se puede ver la planificación del proyecto, teniendo en cuenta la duración de cada tarea y los hitos propuestos.

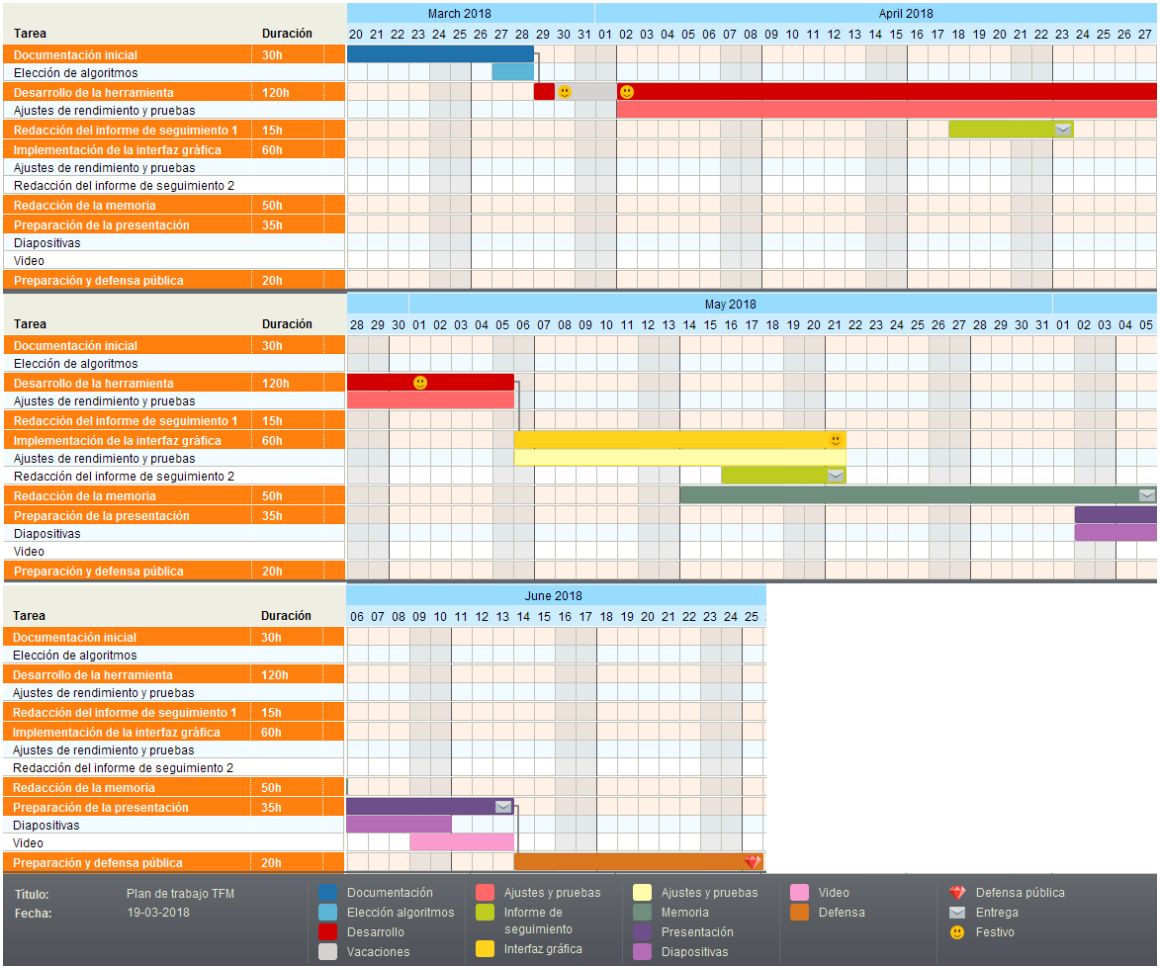

Figura 1.1: Diagrama de Gantt donde se especifica la calendarización de cada una de las tareas, con su duración y sus relaciones de dependiencia (líneas conectoras). Diagrama generado con la herramienta *Tom's planner* (https://www.tomsplanner.com/).

1.4.3. Hitos

Los hitos parciales que se han ido alcanzando en cada una de las Pruebas de Evaluación Continua (PEC) son los siguientes:

Hito	Fecha
Definición de los contenidos del trabajo (PEC0)	05/03/2018
Documentación inicial y plan de trabajo (PEC1)	19/03/2018
Documentación (continuación), diseño del paquete `methylearning`, fase inicial del desarrollo e informe de seguimiento 1 (PEC2)	23/04/2018
Fase final del desarrollo del paquete `methylearning`, desarrollo de la aplicación web asociada e informe de seguimiento 2 (PEC3)	21/05/2018
Redacción de la memoria (PEC4)	05/06/2018
Preparación de la presentación (PEC5a)	13/06/2018
Defensa pública (PEC5b)	25/06/2018

Cuadro 1.2: Relación de hitos con sus fechas de entrega.

1.5. Breve sumario de productos obtenidos

Los productos obtenidos durante la realización de este TFM son los siguientes:

1. **Paquete `methylearning`:** este paquete de `R` implementa un *framework* completo con el que poder aplicar multitud de algoritmos de *feature selection* y de clasificación, a partir de datos de metilación del DNA. El paquete proporciona herramientas para aplicar todas las combinaciones deseadas de estos y evaluar numérica y gráficamente los resultados obtenidos por cada combinación.

2. **Aplicación web asociada:** desarrollada usando *Shiny*, esta aplicación proporciona una interfaz gráfica para el uso de las funcionalidades más relevantes del paquete `methylearning`. Los resultados obtenidos pueden descargarse como archivos separados, además de ser visualizados en el navegador.

Ambos productos están públicamente disponibles a través de sendos repositorios de *bitbucket*, pueden consultarse en las siguientes URLs:

- Paquete `methylearning`:
  `https://bitbucket.org/dpv-dev/methylearning.git`
- `methylearning_App` (aplicación *Shiny*):
  `https://bitbucket.org/dpv-dev/methylearning_app.git`

Ambos se pueden descargar fácilmente yendo a la sección "Downloads" del repositorio o empleando los siguientes comandos en el terminal de Linux (requiere tener `git` instalado):

```
# Para descargar la carpeta del proyecto methylearning:
git clone https://bitbucket.org/dpv-dev/methylearning.git
# Para descargar la carpeta del proyecto methylearning_App
git clone https://bitbucket.org/dpv-dev/methylearning_app.git
```

Para facilitar la instalación y puesta a punto de ambos recursos, se facilita un archivo `README.md` para cada proyecto, además de abundante documentación.

1.6. Breve descripción de los otros capítulos de la memoria

El resto de la memoria está escrita siguiendo el formato de artículo científico. La primera parte es una introducción que pone en contexto, tanto los algoritmos usados en el paquete, como los datos de entrada empleados y las preguntas que se pretende responder usándolos en conjunción con técnicas de *machine learning*. Seguidamente, se incluye una sección de "métodos", con los aspectos más relevantes del diseño y desarrollo del paquete *methylearning* y su aplicación *Shiny*. Más adelante, a modo de "Resultados", se expone un caso práctico de uso, empleando datos públicamente disponibles, para ilustrar las funcionalidades del paquete y los resultados que de él se obtienen. Finalmente, se incluye una discusión de los resultados obtenidos.

Capítulo 2

El paquete `methylearning` y su aplicación *Shiny*

2.1. Introducción

2.1.1. La cromatina, epigenética y salud

La cromatina es un complejo formado por diversos tipos de macromoléculas, principalmente DNA y proteínas, localizado en el núcleo de las células eucariotas. Su estructura, a grandes rasgos, está compuesta por un tipo específico de proteínas, las histonas, que se asocian para formar una estructura conocida como nucleosoma. Este, se une a la doble hebra de DNA, compactándola y organizándola en mayor o menor medida (ver Figura 2.1, extraída de [10]). Las funciones de la cromatina, como estructura que contiene el material genético de las células, son muchas [11]. Entre las más importantes cabe destacar:

1. Proporcionar una forma de compactación del material genético para que pueda ser contenido en el interior del núcleo celular. Esta compactación, resulta muy dinámica y especialmente necesaria durante la división celular, asegurando su integridad.

2. Proteger el material genético de agentes, tanto físicos como químicos, que pueden dañarlo.

3. Contribuir a la regulación génica y de la replicación del DNA, tanto por su propia estructura, que facilita en mayor o menor medida estos procesos, como por las modificaciones químicas de la cromatina, que constituyen todo un conjunto de señales reconocidas por las maquinarias de transcripción y de replicación del DNA.

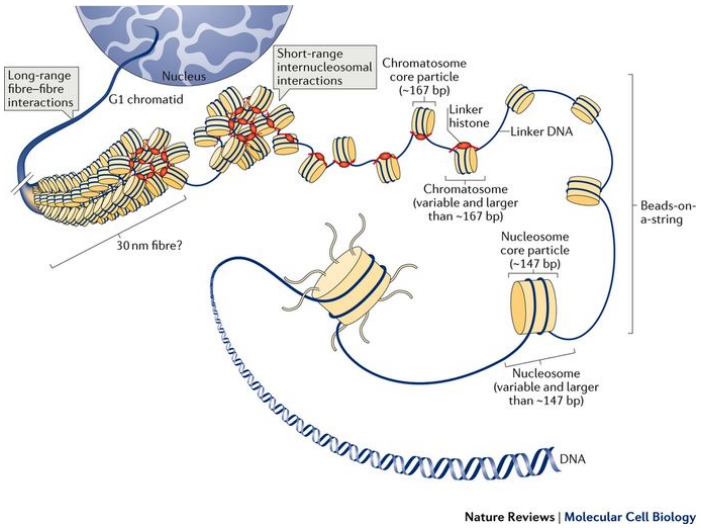

Figura 2.1: Multiples niveles de compactación de la cromatina.

Las modificaciones químicas que se producen, tanto en la molécula de DNA, como en las histonas, se conocen como marcas epigenéticas y son un campo de estudio muy activo en la actualidad. Estas marcas funcionan a modo de señales, que son reconocidas por diversos factores con funciones muy variadas, tales como regulación transcripcional, reparación del DNA, modificación química de la cromatina o funciones estructurales. El patrón de metilación es característico de cada tipo celular y resulta un componente clave para explicar la capacidad de generar tantos tipos celulares diferentes a partir de un mismo genoma. Las marcas epigenéticas son en gran medida heredables, tanto a nivel celular como de organismo, aunque también se ha visto que pueden resultar relativamente dinámicas. En este sentido, se ha observado que el patrón de metilación puede verse afectado, entre muchos otros factores, por el ambiente, el grado de diferenciación celular o la edad del organismo [12, 13].

Se conocen multitud de modificaciones de las histonas, siendo estas de naturaleza química muy variada, como por ejemplo la metilación, acetilación, fosforilación, sumoilación, ubiquitinación, etc. [14]. Por contra, el número de modificaciones de la molécula de DNA que actúan como marcas epigenéticas es mucho menor, destacando especialmente la metilación del DNA. Esta modificación normalmente es una 5-metilcitosina (5mC), ya que el grupo metilo se añade al quinto carbono de una citosina que normalmente se encuentra unida a una guanina mediante un enlace fosfato, lo que se conoce como dinucleótido CpG (citosina-enlace fosfato-guanina) [15].

La 5mC, cuando se da en las regiones reguladoras de los genes, se asocia gene-

ralmente a un estado transcripcional reprimido [16]. Se puede dar de forma más o menos aislada o en agrupaciones del dinucleótido CpG, conocidas como islas CpG. Más recientemente, también se ha asociado la metilación del DNA a un estado transcripcional activo, sobre todo cuando esta se da en el cuerpo del gen o en el extremo 3' UTR (untranslated region) del mismo [17].

El patrón de metilación del DNA está relacionado con multitud de patologías. Entre ellas, destacan un gran número de tipos diferentes de cáncer, enfermedades autoinmunes y neurodegenerativas, entre muchas otras [18]. Se ha comprobado que el nivel de metilación de las células afectadas es diferente al de las células no afectadas, por lo que la determinación de este patrón proporciona una herramienta muy valiosa para el diagnóstico y diseño de tratamientos específicos [19].

2.1.2. Técnicas epigenómicas

El gran interés de la comunidad científica en el estudio de las modificaciones del DNA ha llevado al desarrollo de un gran número de técnicas experimentales diferentes. En los últimos años, aprovechando del auge de técnicas genómicas de alto rendimiento, tales como los *microarrays* y la secuenciación paralela masiva, se han desarrollado todo un conjunto de técnicas de alto rendimiento para el estudio de la epigenética lo que, por analogía, ha llevado a acuñar el término "epigenómica".

De entre todas las técnicas epigenómicas disponibles en la actualidad para determinar el estado de metilación del DNA, destacan especialmente dos grandes grupos: las basadas en la tecnología de *microarrays* y las basadas en secuenciación de alto rendimiento de DNA tratado con bisulfito.

Tecnología de *microarrays* basada en *beads* de Illumina

Desde hace algún tiempo, prácticamente el único fabricante de *microarrays* para el estudio de la metilación del DNA es Illumina [20]. Este basa su tecnología en la disposición de multitud de microscópicas bolas (*beads*) a las cuales se les ha unido previamente las sondas (oligonucleótidos de unos 50 nucleótidos) complementarias a una localización del genoma potencialmente metilada. El DNA de la muestra a analizar se trata con bisulfito sódico, lo que provoca la deaminación de las citosinas, que pasan a convertirse en uracilos. Las citosinas que estuvieran metiladas no sufren esta deaminación y, por tanto, continúan intactas. El DNA tratado de este modo se hibrida con las sondas del *microarray* las cuales, mediante la adición de un par fluoróforos diferentes, son capaces de indicar si la molécula que se ha unido posee una citosina o una timina, en la posición interrogada. Así, comparando la intensidad de fluorescencia de uno y otro fluoróforo (dependiente de la cantidad de moléculas que se hibridaron) se puede determinar el nivel de metilación. Un esquema del

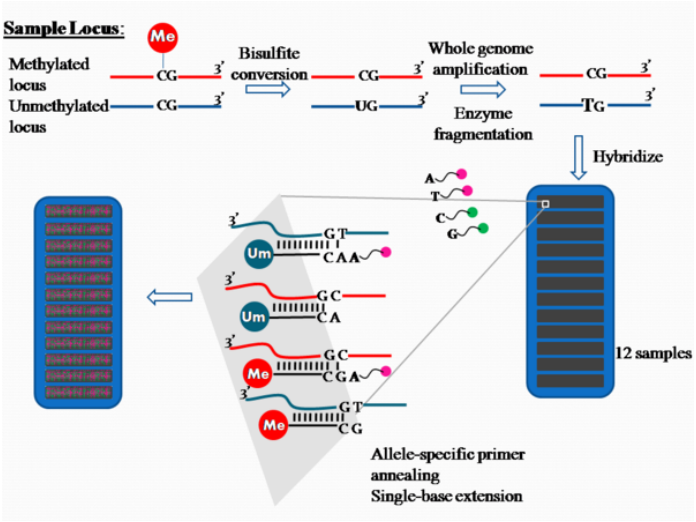

Figura 2.2: Funcionamiento de los *microarrays* de metilación de Illumina.

funcionamiento de esta técnica puede verse en la figura 2.2, extraída de Wikipedia (https://en.wikipedia.org/wiki/Illumina_Methylation_Assay). En esencia, el lector de *microarrays* genera dos valores de intensidad para cada sonda, uno para cada fluoróforo, que indican la cantidad de hibridación que se ha dado tanto de DNA metilado como de DNA no metilado. A partir de estos valores, se calcula el valor beta (β *value*) para cada sonda, a partir de la siguiente fórmula [21]:

$$\beta_i = \frac{M_i}{U_i + M_i + \alpha}$$

Siendo β_i el β *value* para una sonda determinada, M_i su valor de intensidad para el fluoróforo que marca el estado metilado, U_i la intensidad del no metilado y, finalmente, el valor α, que acostumbra a establecerse en 100 y que se recomienda añadir para regularizar el β *value* cuando M_i y U_i son bajos. De este modo, los β *values* pueden tomar valores entre 0 y 1, no llegando a estos dos extremos.

Tres modelos concretos de *micrarrays* de metilación del DNA de Illumina son los que prevalecen en las bases de datos:

- Infinium HumanMethylation27 BeadChip el cual, como su nombre indica, dispone sondas para determinar la metilación de unas 27000 citosinas del genoma humano [22].
- Infinium Human Methylation 450K BeadChip, el más abundante en las bases de datos, posee unas 450000 sondas, conteniendo la mayoría de las sondas de su predecesor, el HumanMethylation27 [23].

- Infinium MethylationEPIC BeadChip, el más moderno de los tres. Disponible desde 2016, contiene sondas para unas 850000 sondas, incluyendo la mayoría de sondas del 27 y del 450k. Es la generación que se encuentra hoy en día disponible en el mercado, estando los anteriores descatalogados [24].

Secuenciación masiva de DNA tratado con bisulfito

Surgidas a partir del desarrollo de las técnicas de *Next Generation Sequencing* (NGS, secuenciación masiva), estas técnicas aprovechan la capacidad actual de secuenciación masiva paralela para determinar el estado de metilación con una resolución a nivel de nucleótido. Entre las diversas variaciones de esta técnica, el ejemplo más clásico sería la técnica de *Whole Genome Bisulfite Sequencing* (WGBS), también conocida como Bisulfite-seq (BS-seq) [25]. Este método consiste en el tratamiento del DNA genómico con bisulfito sódico y su posterior secuenciación masiva. Las citosinas no metiladas se leen como timinas, mientras que la metilación de las citosinas las conserva como tales en la secuenciación. Mediante esta técnica se consigue una alta resolución, abarcando la practica totalidad del genoma.

Al igual que las técnicas basadas en *microarrays*, el WGBS tiene sus ventajas y sus limitaciones. En la tabla 2.1 se resumen las fortalezas más importantes tanto de las técnicas de *microarrays* como del WGBS.

Ventajas *microarrays*	Ventajas WGBS
Coste por muestra reducido, lo que permite estudios más amplios con cohortes más numerosas.	Posibilidad de descubrir nuevas citosinas metiladas, ya que no está limitado a un set de sondas determinado.
Gran robustez, tanto a nivel experimental como de análisis bioinformático y estadístico.	Resolución a nivel de nucleótido y cobertura de todo el genoma.
La menor cantidad de datos generados redunda en un menor coste de almacenamiento y un tiempo de análisis reducido.	Capacidad de extraer información complementaria, como por ejemplo variaciones en la secuencia nucleotídica (*single nucleotide variations*, SNVs).

Cuadro 2.1: Ventajas de los *microarrays* de metilación frente a las ventajas del WGBS.

Además del WGBS, se han desarrollado varias técnicas relacionadas, que pretenden mejorar esta técnica en algún aspecto. Así, entre otras muchas variaciones, se puede nombrar el *Reduced Representation Bisulfite Sequencing* (RRBS) [26], que combina el uso de encimas de restricción, junto con la secuenciación de bisulfito, para enriquecer en secuencias con un alto contenido en CpG, mejorando la resolución y facilitando el análisis. Otro ejemplo podría ser la técnica de *methylated DNA immunoprecipitation* (MeDIP or mDIP) acoplada a NGS [27], lo que se denomina

MeDIP-seq. Esta técnica también produce un enriquecimiento de las citosinas metiladas, ya que se basa en la inmunoprecipitación de la 5mC mediante anticuerpos específicos.

Independientemente de la técnica de secuenciación de bisulfito empleada, los datos que de ellas se obtienen son las lecturas del secuenciador (*reads*). El método de análisis de estos datos acostumbra a incluir una fase de alineamiento al genoma para determinar qué citosinas han cambiado a timina. De este modo se obtiene el número de *reads* correspondientes a citosinas metiladas y no metiladas, para cada CpG secuenciada. A partir de estos valores, empleando la siguiente fórmula, se estima el ratio de metilación, conceptualmente análogo al β *value* de los *microarrays* de metilación:

$$R_i = \frac{M_i}{M_i + U_i}$$

Siendo R_i el ratio de metilación en la posición "i", M_i el número de *reads* metilados en la posición "i" y U_i el número de *reads* no metilados en la posición "i". Este valor puede variar entre 0 y 1 y, en contraposición al β *value*, sí puede ser 0 y 1, para el caso en que todos los *reads* sean no metilados o metilados, respectivamente.

2.1.3. *Machine learning* y epigenética

Dado que el epigenoma tiene un papel fundamental en los procesos que llevan a las células a adoptar diferentes fenotipos, al mismo tiempo que proporciona cierta capacidad de respuesta a estímulos del microambiente celular, predecir las regiones susceptibles a cambios epigenéticos resulta clave a la hora de comprender multitud de fenómenos biológicos, entre los que destacan ciertas enfermedades. Debido a su estrecha relación con múltiples aspectos biológicos, la epigenética presenta un gran potencial para ser usada en conjunción con técnicas de ML. Así, en la literatura se encuentran ejemplos muy variados del uso de ML a partir de datos epigenéticos y epigenómicos. Así, el ML se ha empleado para la predicción de marcas epigenéticas [28], para el estudio de la proporción de diversos tipos celulares en una muestra [29], para predecir parámetros biológicos muy variados como por ejemplo la edad [30] y para la detección de ciertos condicionantes ambientales [31].

Una de las sinergias más prometedoras entre epigenética y ML, es su aplicación con fines diagnósticos y de prevención. En los últimos tiempos, es muy destacable el esfuerzo que varios consorcios han estado realizando para dotar a la comunidad científica de datos epigenómicos, acompañados de datos de otras muchas técnicas ómicas, además de datos clínicos, con el fin de mejorar la prevención, el diagnóstico y el tratamiento de ciertas enfermedades. En este sentido, destaca especialmente el proyecto TCGA (https://cancergenome.nih.gov/), centrado en ofrecer múltiples

tipos de datos ómicos para un conjunto de tipos de cáncer. Aunque no específicamente enfocado a patologías, también merece la pena destacar el proyecto ENCODE (*Encyclopedia of DNA Elements*), que también ofrece multitud de datos ómicos de muy diversa índole. Estos proyectos, junto con el esfuerzo de muchos otros laboratorios han sido y están siendo claves para propiciar el uso de técnicas de ML, altamente dependientes del número de muestras, en el campo de la epigenómica [32–35].

Para el diagnóstico y otras posibles aplicaciones del ML, la tarea consiste en clasificar un conjunto de muestras en base a unas características. Así, se puede pretender clasificar la muestra entre tumoral y sana, predecir el tejido de origen de una metástasis, o evaluar la respuesta a un cierto tratamiento. Si al modelo de ML se le aportan las etiquetas correspondientes para la correcta clasificación de las muestras de entrenamiento, el modelo tratará de ajustarse a estas etiquetas para realizar sus predicciones. Este tipo de modelos se denominan supervisados, ya que se les aporta la solución "correcta" de antemano, lo que dirige su ajuste. Por contra, si no se aportan las etiquetas de clasificación de las muestras y se emplean algoritmos que tratan de agrupar y clasificar las muestras en base sus características, el modelo se denomina no supervisado [36]. Dado que en las tareas de diagnóstico la mayor fuente de variabilidad entre las muestras no tiene porqué estar relacionada con el tipo de clasificación que se quiera llevar a cabo, los métodos no supervisados suelen mostrar un rendimiento general menor y, dado la importancia de la tarea, normalmente se ha optado por métodos de clasificación supervisados. Sin embargo, existen diversos problemas que surgen a la hora de aplicar estos métodos para tareas de diagnóstico a partir de datos epigenómicos [5]:

1. Covariables con un efecto mucho mayor en la variabilidad de las muestras.
2. Reducido número de muestras y alto número de variables con las que entrenar el modelo.
3. Descompensación, en cuanto a número de muestras disponibles, para las diversas clases que se pretenden predecir.

Ejemplos típicos del problema de las covariables (o factores de confusión) se dan, por ejemplo, cuando la incidencia de una patología es mucho mayor en uno de los sexos o cuando las proporciones de los diversos tipos celulares son muy diferentes entre clases [37]. Es un problema serio ya que, inadvertidamente, podemos estar generando un modelo de clasificación que no prediga en base a los parámetros que se le han indicado.

Para el segundo problema, el de la dimensionalidad, la solución pasa por realizar una selección de marcadores (FS) que consiga reducirlos a los realmente importantes para el problema que se desea tratar.

El tercer problema suele afectar especialmente al poder predictivo para la clase más interesante de las que se pretende predecir. Normalmente, la característica que se quiere predecir con precisión suele ser el estado patológico, del que las muestras acostumbran a escasear, en comparación a las muestras control. Por esta razón, se tiende a entrenar los modelos con muchas muestras control y muy pocas muestras patológicas, lo que reduce la capacidad del modelo de distinguirlas. Existen diversas estrategias para mitigar este problema, aunque lo más efectivo siempre es conseguir más muestras de la clase infrecuente.

2.1.4. Algoritmos de *Feature Selection*

Por las razones expuestas en el apartado anterior, la FS resulta de extrema importancia a la hora de trabajar con datos epigenómicos. Esto ha llevado a que se desarrollen una gran cantidad de métodos de selección de marcadores basados en diversas estrategias. Los algoritmos de FS típicamente se dividen en tres grandes grupos: *Filters*, textitwrappers y *embedded* [38, 39]. Además de estos, también dentro de lo que se podría considerar como FS, está el proceso que se conoce como "extracción de características", más comúnmente denominado reducción de la dimensionalidad. Este proceso pretende extraer la información más relevante de las variables, transformándolas en un conjunto más pequeño de nuevas variables. Estos algoritmos pueden transformar las variables combinando varias de ellas por métodos tanto lineales (como por ejemplo el análisis de componentes principales, PCA) como no lineales (como el algoritmo Isomap [40]). Sin embargo, a pesar de funcionar muy bien para tareas de clasificación, la transformación de las variables hace que sea difícil determinar cuáles han sido más relevantes. En el caso del diagnóstico basado en datos epigenómicos, el conocer qué variables (CpGs, que pueden estar asociadas a ciertos genes) son las más discriminantes, es de gran relevancia para poder entender el modelo y los mecanismos biológicos subyacentes, además de tener un gran valor en la búsqueda de biomarcadores. Por estas razones, las técnicas de reducción de la dimensionalidad no son tan comúnmente usadas para datos epigenómicos.

Algoritmos de filtrado de variables (*filters*)

Desde el surgimiento de los primeros *microarrays* de expresion, la FS se hizo prácticamente imprescindible a la hora de aplicar algoritmos de clasificación sobre este tipo de sets de datos. Dados los recursos computacionales disponibles, los primeros métodos en aplicarse fueron los *filters* y todavía hoy se emplean de . Estos métodos actúan como preprocesado de los datos, de forma independiente al algoritmo de clasificación posterior. Se pueden distinguir dos tipos diferentes:

Univariantes Tienen en cuenta cada variable de manera aislada. Como ejemplos de este tipo de *filters* se pueden citar:

- Filtrado por significación en test estadístico, como por ejemplo un ANOVA (análisis de la varianza), comparando las diferentes clases.
- *Information Gain* (ganancia de información), es uno de los métodos más utilizados. Las variables se ordenan en función de cuanta información proporcionan con respecto a la clasificación deseada. Se pueden usar múltiples medidas de la información aportada por cada variable, la más común de las cuales es la entropía de Shannon [41].

Multivariantes Tienen la ventaja de evaluar las variables en su conjunto, pudiendo modelar las dependencias entre variables, aunque a cambio son más costosos computacionalmente. Como ejemplo se pueden citar:

- *Correlation-based Feature Selection* (CFS), como su nombre indica, se basa en la correlación que existente, tanto entre las variables, como con la clase a predecir. De este modo, se seleccionan las variables menos intercorrelacionadas y con mayor poder predictivo [42].
- ReliefF, el cual es una extensión del algoritmo Relief original [43], funciona comparando el valor de una muestra elegida al azar con su vecino más próximo de su misma clase y de otra clase. Los valores de las variables se comparan y se utilizan para actualizar las puntuaciones de relevancia de cada variable. La idea es que una variable útil debe poder diferenciar entre muestras de diferentes clases y, por lo tanto, sus valores deben ser más parecidos entre muestras de la misma clase. Originalmente diseñado para clasificaciones binarias, ReliefF extiende su uso a múltiples clases [44].
- *minimum Redundancy Maximum Relevance* (mRMR), selecciona variables en base a un criterio dual: máxima relevancia respecto a la clasificación y mínima redundancia con otras variables [45].

Algoritmos basados en el rendimiento de un clasificador (*wrappers*)

En el caso de datos epigenómicos, los algoritmos de filtrado de variables del tipo *wrapper* han sido los que han recibido la menor atención, principalmente debido a su alto coste computacional, que crece exponencialmente con la dimensionalidad. Además, también tienen mayor tendencia al sobreajuste del modelo, problema que se agrava con la escasez de muestras. No obstante, existen ciertos ejemplos de algoritmos que han sido aplicados exitosamente a datos de alta dimensionalidad:

- Algoritmo de selección de variables secuencial, presentados de diversas maneras, como por ejemplo *sequential forward*, que parte de una selección aleatoria de variables sobre las que va añadiendo nuevas variables y evaluando si aportan una mejora en la clasificación; o *sequential backward*, que actúa a la inversa, partiendo del total de variables extrayéndolas de forma secuencial y evaluando su influencia en la clasificación. Este algoritmo es tremendamente costoso a nivel computacional, ya que el número de predicciones del clasificador asociado crece exponencialmente con el número de variables, por lo que los ejemplos de uso con datos de alta dimensionalidad son escasos [46].
- Algoritmo genético, que emplea los principios básicos de los algoritmos genéticos pero como función de *fitness* incorpora el resultado de un clasificador. Son también muy computacionalmente costosos, aunque se pueden encontrar casos de su uso con éxito [47].
- Boruta, implementado en un paquete de `R` homónimo [48], se basa en la eliminación iterativa de variables que aparecen como menos relevantes que variables al azar, usando como criterio de evaluación el rendimiento en un clasificador de tipo *Random Forest*.

Selección de variables incluida en el algoritmo de clasificación (*embedded*)

Como se ha visto, una de las principales desventajas de los *filters* es que no interactúan con el clasificador, con lo que el conjunto de variables seleccionado no está tan específicamente preparado para la tarea de clasificación. Por contra, los *wrappers* pueden obtener mejor rendimientos para tareas de clasificación, a expensas de un coste computacional que puede convertirlos en prohibitivos para algunos tipos de sets de datos, además de acarrear un problema de sobreajuste al modelo empleado para la selección de variables. Una solución intermedia son los métodos de FS del tipo *embedded*, que incorporan el proceso de selección de variables a la misma creación el modelo de clasificación. Como ejemplo de este tipos de algoritmos usdo con datos de alta dimensionalidad está el algoritmo de eliminación recursiva de variables (*recursive feature elimination*)con máquinas de soporte vectorial (*support vector machines*), SVM-RFE [49], que iterativamente entrena una SVM como clasificador con un subconjunto de las variables, de las que elimina la menos relevante según este clasificador.

2.1.5. Algoritmos de clasificación

Como se ha comentado, para el problema de clasificación aplicado al diagnóstico, los algoritmos acostumbran a ser de tipo supervisado. Existe una inmensa

cantidad de algoritmos diferentes. Nombrarlos todos resultaría demasiado extenso y no es el propósito de este texto. Sin embargo, si se comentan los métodos más utilizados para datos epigenómicos, que son los que se han incluido en el presente proyecto.

k-Nearest Neighbor

El algoritmo k-NN es un clasificador muy popular basado en *nearest neighbor* (vecino más cercano) [50]. Es uno de los algoritmos de ML más simples. Estrictamente hablando, está en la frontera de lo que se puede considerar un algoritmo de aprendizaje automático, ya que la etapa de entrenamiento no sé da propiamente, si no que para cada nuevo caso sin etiquetar (de clase desconocida), selecciona los *k* vecinos más próximos del set de datos de entrenamiento y le asigna la etiqueta mayoritaria. El valor de *k* se debe escoger de antemano, aunque se suele empezar por valores de $k = \sqrt{n}$, siendo n el número de muestras del set de datos de entrenamiento. También existen varias opciones para calcular la distancia entre vecinos. Una opción muy utilizada es la distancia Euclídea. Ya que esta medida de distancia no tiene en cuenta las diferencias de escala, conviene normalizar los datos antes de ser usados por el algoritmo.

A pesar de su simpleza, resulta un algoritmo muy efectivo en ciertas situaciones y un buen punto de partida para muchos proyectos. Además, al ser una técnica no paramétrica, no presupone ninguna distribución concreta para los datos subyacentes. Por contra, acostumbra a requerir cierto ajuste del parámetro *k*, además de no aceptar datos de tipo no numérico.

Red Neuronal Artificial

El algoritmo de Redes Neuronales Artificiales (ANN de sus siglas en inglés, *Artificial Neural Networks*) modela las relaciones entre un conjunto de señales de entrada y un conjunto de señales de salida, usando un modelo basado en una red de nodos (o neuronas artificiales) cuyo funcionamiento está inspirado en el funcionamiento del cerebro [51]. Estas redes están formadas por diferentes capas y cada una de ellas por uno o varios nodos. Las señales que recibe cada nodo son integradas por medio de una **función de activación** que determina la señal que se envía a los nodos de la siguiente capa. La importancia relativa de cada una de las señales de entrada es modulada mediante *pesos*. Estos pesos son modificados en cada uno de los pasos de entrenamiento hasta resultar en una respuesta que genere una señal de salida adecuada. La primera capa de esta red está constituida por los nodos de entrada (*input nodes*) cuyo número viene determinado por las variables existentes en los datos de entrada. La última capa está constituida por uno o varios nodos

de salida, que corresponden a los resultados de la predicción o clasificación. Entre estas dos capas pueden existir una o más capas, denominadas capas ocultas (*hidden layers*) constituidas por uno o más nodos, denominados nodos ocultos (*hidden nodes*). Aunque redes de una única capa son capaces de modelar ciertos problemas, se requieren redes multicapa para modelar problemas complejos. Las redes neuronales con más de una capa oculta a menudo se denominan redes neuronales profundas (*deep neural networks*) y el entrenamiento de tales redes se denomina *deep learning*.

Existen múltiples tipos de redes neuronales, dependiendo de la dirección que siguen las señales y de las conexiones existentes entre los nodos. Uno de los más comunes es el constituido por varias capas en las que la señal se propaga en una dirección, de una capa a la siguiente, hasta llegar a la capa de salida (*Multilayer Perceptron*, **MLP**) [36].

A pesar de los numerosos tipos de ANN, cada una puede ser definida por las siguientes características:

- Una **función de activación**, que transforma las señales de entrada de cada neurona en una única señal de salida que será enviada a la siguiente capa de la red.
- La **topología de la red**, que describe el número de neuronas en el modelo, así como el número de capas y la maneras en la que estas están interconectadas.
- El **algoritmo de entrenamiento**, que especifica como se definen los pesos de cada conexión, para excitar o inhibir las neuronas artificiales en proporción a la señal de entrada.

Debido a su gran flexibilidad y capacidad para modelar problemas muy complejos, las ANN han sido aplicadas con éxito a multitud de problemas de campos tan diversos como el reconocimiento del lenguaje hablado y escrito, reconocimiento de imágenes, automatización de sistemas (por ejemplo, conducción autónoma de vehículos) y modelización de sistemas complejos, tales como patrones climáticos, dinámica de fluidos y un largo etcétera de otros fenómenos científicos, sociales o económicos.

Entre las fortalezas de este conjunto de algoritmos destacan su capacidad para modelar problemas muy complejos, obteniendo un buen rendimiento en un amplio espectro de casos. Por contra, resultan computacionalmente muy costosas, a la vez que muy dadas al sobreajuste. Sin embargo, uno de los mayores problemas de las ANN a la hora de aplicarlas en el campo de la salud es que el modelo resultante es de una complejidad muy grande, y muy difícil de interpretar. Por esta razón, a los modelos generados con ANN se les denomina modelos de caja negra (*black box*).

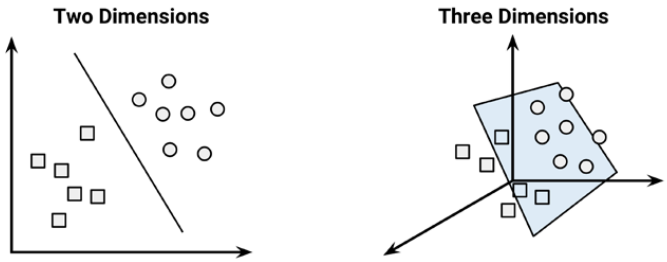

Figura 2.3: Representación de un hiperplano.

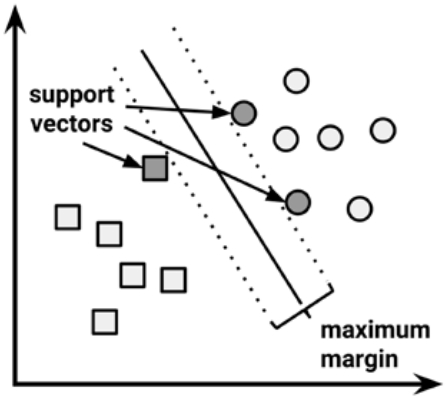

Figura 2.4: MMH y vectores de soporte.

Máquinas de soporte vectorial

El algoritmo de máquina de soporte vectorial (*Support Vector Machine*, SVM) puede ser usado para la resolución de problemas tanto de clasificación como de regresión. Combina aspectos de los algoritmos de aprendizaje por *nearest neighbors* con métodos de modelización usando regresión lineal. Tiene por objetivo encontrar un tipo de superficie plana multidimensional llamada **hiperplano**, que divide los datos en múltiples clases. Una representación en dos y tres dimensiones se puede ver en la Figura 2.3, extraída de [36].

Por tanto, el algoritmo SVM intenta encontrar el/los hiperplano/s óptimo/s que logren separar los datos de entrenamiento según sus clases, lo que implica la búsqueda del **Hiperplano de Margen Máximo** (*Maximum Margin Hyperplane*, **MMH**). Encontrando el MMH se consigue que el modelo generalice mejor para datos futuros y que sea más insensible al ruido que pueda existir en los datos. La búsqueda del MMH pasa por encontrar los **vectores de soporte** (*support vectors*), que son los puntos de cada clase más cercanos al MMH, indicados con flechas en la siguiente Figura 2.4, extraída de [36].

El algoritmo SVM, tal cual se ha explicado, funciona para datos linealmente separables. Sin embargo, este algoritmo también se puede usar cuando los datos no son linealmente separables. Para esos casos, la solución puede ser añadir una **slack variable** (variable de holgura) al modelo, que permita que ciertos puntos caigan en el lado incorrecto del MMH. Una variable de coste (C) se aplica a todos los puntos que caigan en el lado incorrecto del MMH y, por tanto, el algoritmo intentará minimizar el coste total, en lugar de encontrar el MMH. Además de la **slack variable**, otra solución al problema de la no linealidad de los datos es utilizar el **kernel trick**. Este consiste en añadir dimensiones adicionales a los datos para que puedan ser separados linealmente en estas nuevas dimensiones. En esencia, consiste en crear nuevas variables a partir de relaciones matemáticas entre las variables de partida de los datos. Así, muchos algoritmos **SVM** permiten elegir entre varios **kernels**. Entre muchos otros, podemos destacar:

- **Kernel lineal**, que no transforma los datos.
- **Kernel polinómico**, que añade una simple transformación no lineal a los datos.
- **Kernel sigmoide**, que da como resultado un **SVM** análogo a una red neural artificial que use una función de activación sigmoide.
- **Kernel RBF Gaussiano**, RBF por *Radial Basis Function*, que da como resultado un **SVM** análogo a una red neural artificial RBF. El kernel RBF funciona bien en muchos casos reales, por eso se considera un buen punto de partida.

Este algoritmo supervisado destaca por su precisión en muchos tipos de escenarios diferentes, viéndose poco afectado por el sobreajuste o los datos con mucho ruido. Sin embargo, puede resultar lento en su fase de entrenamiento, además de presentar el mismo problema de interpretación del modelo que las ANN, siendo también considerados modelos de caja negra.

Árboles de decisión

Los árboles de decisión (*decision trees*) son clasificadores que realizan complejas tareas de clasificación basándose en una estructura de árbol, que establece relaciones entre las variables explicativas y las clases a predecir. A partir del total de los datos, se van estableciendo sucesivas subdivisiones basadas en simples criterios, de manera análoga a como, a partir del tronco de un árbol, van apareciendo ramas más estrechas que se van subdividiendo en ramas todavía más estrechas. Así, el caso a evaluar empieza en un nodo raíz, a partir del cual va pasando por los nodos

de decisión que, basados en criterios dados por las variables explicativas, conducen a los nodos terminales o nodos hoja, que determinan la acción a tomar, es decir, la clase asignada al caso.

La estructura tipo diagrama de flujo que poseen estos árboles de clasificación no sólo es útil para el funcionamiento interno del clasificador, también puede ser fácilmente entendible por una persona, lo cual brinda una potente herramienta para investigar como ha funcionado el modelo y porqué ha funcionado o no, para cierta tarea. Esto los hace especialmente adecuados para aplicaciones que requieran de total transparencia, como por ejemplo el diagnóstico médico.

Los árboles de clasificación se construyen usando la heurística llamada partición recursiva, también conocida como estrategia *divide and conquer* (divide y vencerás). Esta técnica se basa en dividir los datos sucesivamente en subgrupos más pequeños hasta que los subgrupos son suficientemente homogéneos u otro criterio de parada se alcanza.

Entre las numerosas implementaciones de árboles de clasificación, el algoritmo C5.0 (http://www.rulequest.com/see5-info.html), desarrollado por J. Ross Quinlan [52], destaca por ser una de las más conocidas, habiéndose convertido en estándar. Sin entrar en la matemática subyacente, este algoritmo funciona de manera que cada subdivisión reduzca la entropía, es decir, las subdivisiones resultantes generen grupos más homogéneos (que pertenezcan mayoritariamente a una sola clase). El balance ente el sobreajuste y el poco ajuste del modelo es algo complicado de obtener en este tipo de clasificadores. Precisamente, una de las ventajas de la implementación del algoritmo C5.0 es lo sencillo que resulta ajustar sus opciones de entrenamiento, lo que ayuda en este proceso de ajuste del modelo.

Entre las ventajas de los árboles de decisión destacan su buen rendimiento para la mayoría de problemas, su capacidad para aceptar cualquier tipo de datos, su método implícito de selección de variables y lo útiles que resultan para proporcionar modelos fácilmente explicables. Por contra, tienen mucha tendencia al sobreajuste y, los árboles muy extensos, pueden resultar difíciles de interpretar, efectuando subdivisiones contraintuitivas en algunos casos.

Random Forest

El algoritmo Random Forests (o *decision tree forests*), desarrollado por Leo Breiman [53], es una aproximación de *meta-aprendizaje* (*meta-learning*) que utiliza el método de *ensemble*. Este método se basa en la idea de que, combinando varios clasificadores de pobre rendimiento, se obtiene un clasificador de mejor rendimiento. Así, el método Random Forests construye un clasificador mejor a base de juntar un grupo (*ensemble*) de árboles de decisión. Combina los principios básicos de la técnica de *bagging* (también llamada *bootstrap aggregating*), por la cual se generan

un número de sets de datos de *training* usando muestreo por *bootstrap* de los datos de *training* originales, con selección aleatoria de variables (del total de variables explicativas), para añadir más diversidad a los árboles de clasificación. Posteriormente, se agrupan estos árboles de clasificación formando el "bosque" (*forest*). La clase final se asigna por un sistema de voto entre los árboles generados.

Como método de *ensemble* que es, utiliza sólo una parte pequeña y aleatoria del conjunto de variables explicativas en cada modelo. Por ello, puede manejar sets de datos extremadamente grandes, como los datos generado por técnicas epigenómicas. Al mismo tiempo, sus ratios de error para la mayoría de tareas de aprendizaje están a un nivel puntero.

Glmnet

El algoritmo glmnet, incluido en el paquete de `R` homónimo, realiza las tareas de regresión y clasificación ajustando un modelo lineal generalizado vía máxima verosimilitud penalizada (*penalized maximum likelihood*). La regularización, el proceso matemático para reducir el sobreajuste del modelo, se calcula mediante penalización "lasso" (*least absolute shrinkage and selection operator*) o "elasticnet". Es un algoritmo muy eficiente computacionalmente y que puede aplicar varios tipos de regresión: lineal, logística, multinomial, de *poisson* y regresión Cox [54].

LDA

El algoritmo LDA (*Linear Discriminant Analysis*) está basado en el discriminante lineal de Fisher, publicado en 1936. Generalmente se emplea como técnica de reducción de la dimensionalidad, aunque su uso también se puede extender a tareas de clasificación. El objetivo es proyectar el set de datos a un espacio de dimensionalidad menor con buena separabilidad de las clases para reducir el sobreajuste, al tiempo que se reduce el coste computacional. El discriminante lineal de Fisher se formuló para problemas con dos clases [55], que posteriormente se generalizó para su uso en problemas con más clases [56].

Su enfoque es muy similar al análisis de componentes principales pero, además de encontrar los ejes que maximizan la varianza de los datos, también se buscan los ejes que maximizan la separación entre las clases. Además, a diferencia del PCA que puede ser considerado como un algoritmo no supervisado, el LDA es un algoritmo supervisado.

2.1.6. Validación del modelo

La estimación del rendimiento de un modelo a partir de los datos de entrenamiento acostumbra a ser muy optimista. Los errores de clasificación que se dan en los mismos datos de entrenamiento (error de resubstitución) pueden ser usados para un primer diagnóstico y descartar muestras o modelos de clasificación de bajo rendimiento. No obstante, el error de resubstitución no es un buen indicativo del rendimiento que tendría el modelo con datos diferentes a los de entrenamiento. Esto es principalmente debido al sobreajuste del modelo a los datos de entrenamiento. Este sobreajuste siempre está persente, en menor o mayor medida dependiendo de la estrategia seguida, pero inevitable en última instancia. Así, en lugar de confiar en el error de substitución, la mejor manera de evaluar el rendimiento de un modelo es utilizarlo con datos nuevos.

Una posible estrategia es dividir los datos iniciales en datos de entrenamiento y datos de test. Sin embargo, esta estrategia tiene el inconveniente de reducir el set de datos de entrada, lo que en el caso de sets de datos con pocas muestras, puede resultar muy perjudicial para el rendimiento del modelo de clasificación. Afortunadamente, se han desarrollado otros métodos de estimar el error que tendría el modelo al ser usado con otros datos. A continuación se muestran las estrategias más utilizadas en este sentido, para una revisión más exhaustiva, consultar [36].

Cross-validation

La validación cruzada (*cross-validation*, CV) consiste en dividir los datos en un número determinado de divisiones o *folds* (denominado con la letra k). Posteriormente, se procede a entrenar el modelo, de manera que los datos contenidos en cada *fold* son usado como datos de validación (test) de un modelo entrenado con el resto de los datos. Finalmente, se calcula la media aritmética obtenidas de las medidas de evaluación en las diferentes particiones.

El número k de *folds* puede ser cualquiera, aunque usualmente se escoge 10, ya que empíricamente se ha visto que aumentar el número de divisiones a más de 10 no mejora el modelo significativamente. Un caso extremo sería el uso de un k tan grande como el tamaño de la muestra, lo que se conoce como *leave-one-out cross-validation* (LOOCV), que implica separar los datos de manera que en cada iteración haya una muestra para los datos de test y el resto para el entrenamiento. A pesar de presentar ciertas ventajas, el alto coste computacional de esta estrategia ha limitado su uso real.

Una variación de esta técnica sería la validación cruzada repetida (*repeated* CV). Como su nombre indica, consiste en realizar la *k-fold CV* un número determinado de veces y promediar los resultados. Una estrategia muy utilizada es realizar *10x*

repeated 10-fold CV (10 repeticiones de 10-*k* validaciones cruzadas) que, aunque muy intensivo a nivel computacional, proporciona unos resultados muy robustos.

Bootstrap

Una estrategia de validación ligeramente menos utilizada que la CV es el método de *boostrap*. Usando este método, se generan un número determinados de sets de datos de entrenamiento y de test al azar, que luego se utilizan para estimar el rendimiento del modelo, promediando los resultados de cada etapa de *boostrap*. De este modo, la diferencia principal con la CV es que, mientras que en la CV cada muestra aparece una sola vez en cada iteración, por el método de *boostrap* las muestras se seleccionan al azar **con reemplazamiento** para generar el set de datos de entrenamiento, eligiendo las muestras que no entraron en el set de datos de entrenamiento para formar el set de test. De este modo, en el set de datos de entrenamiento, existen muestras repetidas en cada iteración. Esto hace que el método de *boostrap* tienda a rendir menos que la CV. No obstante, este método está especialmente indicado para sets de datos pequeños, en los que sí tiende a rendir algo mejor.

2.2. Métodos

2.2.1. Diseño del paquete `methylearning`

Desde el principio del proyecto, por la practicidad para su distribución, se decidió que el software tendría forma de paquete de `R`. Dado que las dos características clave que se decidió implementar, FS y clasificación, pueden ser fácilmente compartimentadas, se optó por un diseño modular, basado en la interacción de tres objetos de `R` (del tipo S3). Este diseño modular facilita la revisión y ampliación del código. Las tres clases principales que implementa el paquete `methylearning` son las siguientes:

1. `ml_data`: es el objeto que acepta y formatea los datos de entrada que proporciona el usuario y lo prepara para los siguientes objetos.

2. `ml_fs`: es durante la creación de este objeto, a partir del objeto `ml_data`, cuando se da la etapa de FS especificada por el usuario. Esta clase de objeto es una clase derivada de la clase `ml_data`, por lo que hereda sus métodos.

3. `ml_cls`: durante la creación de este objeto a partir de un objeto `ml_fs` se da la etapa de clasificación, a partir de los conjuntos de marcadores seleccionados previamente.

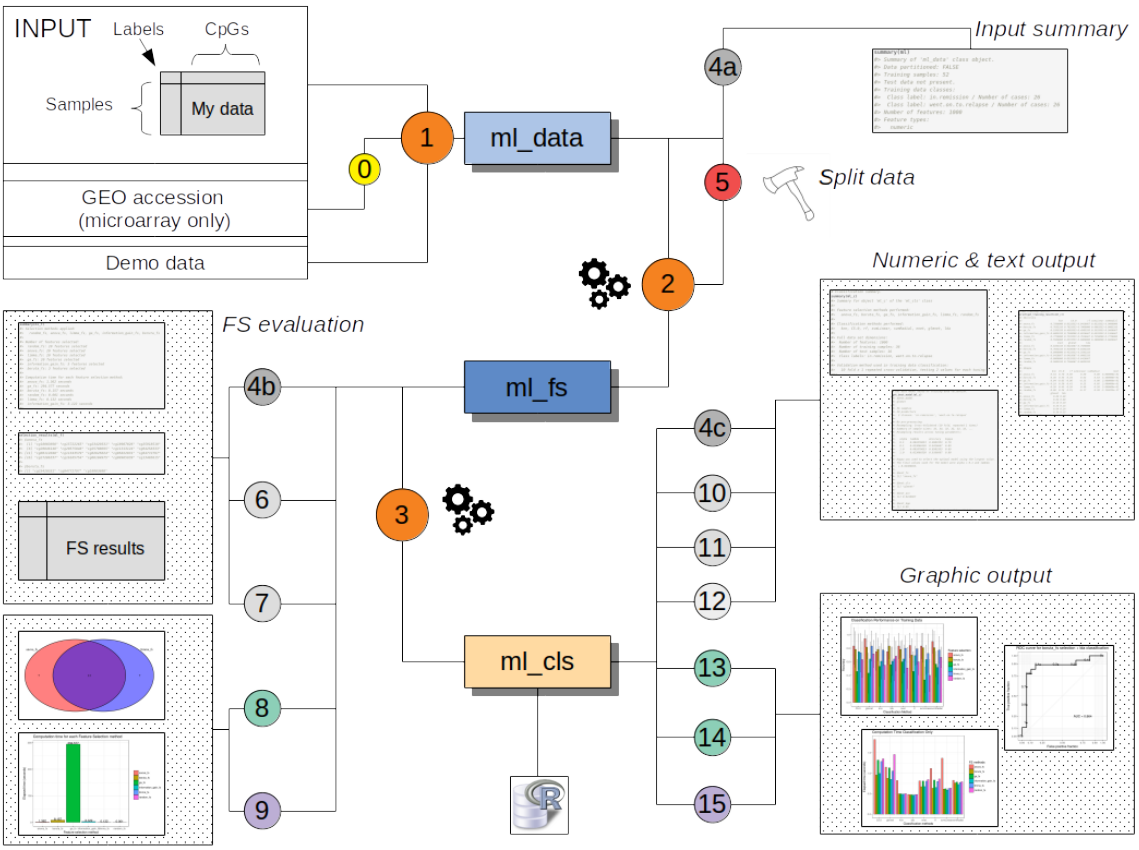

Figura 2.5: Esquema de funcionamiento del paquete *methylearning*.

Por lo tanto, el análisis que permite llevar a cabo este paquete está planteado de manera secuencial, con una primera etapa de entrada y preparación de los datos, una segunda etapa en la que se realiza la FS y se pueden evaluar los resultados y, finalmente, una tercera etapa en la que se realiza la clasificación, que culmina en la detallada evaluación de los resultados obtenidos con los diferentes conjuntos de marcadores seleccionados y algoritmos de clasificación aplicados. Cabe destacar que, dado el gran gasto computacional de algunos de los algoritmos implementados, el paquete está paralelizado en múltiples puntos críticos para mejorar la velocidad de ejecución.

En la figura 2.5 se puede ver un esquema general de los componentes más destacados de paquete `methylearning`, indicando el flujo de trabajo típico que se puede llevar a cabo.

A continuación se detallan cada una de las clases implementadas en el paquete y sus métodos y características más relevantes.

Clase `ml_data`

Es la clase que se encarga de aceptar los datos de entrada y prepararlos para el resto del análisis. Como datos de entrada acepta cualquier `data.frame`, con una dimensión mínima de 2x2 en los que todos sus valores sean numéricos, a excepción de la columna que se indique como `labels_column`, que puede ser de cualquier tipo convertible a `factor`. Todo el paquete se ha diseñado teniendo en cuenta las peculiaridades de los datos generados por experimentos *genome wide* de metilación del DNA: *microarrays* de metilación de Illumina y secuenciación masiva de bisulfito. Para el primer caso se recomienda el uso de los valores β (β-*values*) y para el segundo se recomienda usar el ratio de metilacíon (*methylation ratio*). Sin embargo, cualquier tipo de dato numérico comprendido entre 0 y 1 debería poder ser usado por el paquete.

Las filas han de corresponder a las muestras (se recomienda nombrar las filas con los nombres de muestra) y las columnas han de ser las variables o marcadores (las sondas en los *arrays* o posiciones en los experimentos de secuenciación). Una de las columnas, como se ha comentado, ha de corresponder a las etiquetas de clase de las muestras, que se usarán para su clasificación. Por defecto, la primera columna se toma como la columna de las etiquetas de clase.

Se han implementado dos funciones para crear objetos de la clase `ml_data`:

- La función constructora `ml_data`, que acepta un `data.frame` de las características indicadas (indicada por el círculo naranja "1" en la figura 2.5).
- La función `get_GEO_methylarray` (indicada por el círculo amarillo "0" en la figura 2.5), que acepta un objeto de la clase `GSE`, generado empleando la función `GEOquery::getGEO`, del paquete `GEOquery` [57]. Mediante este objeto y el nombre de la columna a emplear como etiquetas de clase del `data.frame` `phenoData` del objeto `GSE`, la función `get_GEO_methylarray` extrae los datos de entrada (β values) y los preprocesa, devolviendo un objeto de la clase `ml_data`.

Los objetos de la clase `ml_data` implementan una serie de métodos:

- `summary`, indicado por el círculo gris "4a" en la figura 2.5, proporciona un resumen en texto de los detalles más relevantes de los datos contenidos en el objeto.
- `split_data`, indicado por el círculo rojo "5" en la figura 2.5 este método implementa la posibilidad de dividir los datos en datos de entrenamiento y de test. Esta subdivisión será importante para poder o no acceder a la evaluación del rendimiento a partir de los datos de test, hacia el final del análisis. A pesar

de la división de los datos, la etapa de FS se realiza con el total de los datos, a no ser que se indique lo contrario.

Clase `ml_fs`

Una vez creado el objeto de la clase `ml_data`, este se puede usar con la función `ml_fs`, indicada por el círculo naranja "2" en la figura 2.5. Esta es la encargada de realizar la FS, devolviendo un objeto de la clase `ml_fs`.

Los objetos de la clase `ml_fs` implementan una serie de métodos, que permiten explorar y evaluar los resultados obtenidos en la selección de marcadores:

- `summary`, indicado por el círculo gris "4b" en la figura 2.5, proporciona el resumen en texto específico del objeto `ml_fs`, seguido del resumen de un objeto de tipo `ml_data`, que también es.
- `selection_methods`, que devuelve un `character vector` con los algoritmos de selección de marcadores utilizados.
- `selection_results`, indicado por el círculo gris "6" en la figura 2.5, que devuelve una `list` en la que cada elemento es el resultado de FS de un algoritmo.
- `plot_venn_fs`, indicado por el círculo verde "8" en la figura 2.5, que utiliza el paquete `vennDiagram` para producir diagramas de Venn de 2 hasta 5 selecciones de variables.
- `selection_df`, indicado por el círculo gris "7" en la figura 2.5, que devuelve un `data.frame` con los algoritmos de FS en las columnas y todos los marcadores seleccionados por todos los métodos de FS aplicados como filas. Los valores son *TRUE/FALSE*, dependiendo de si la variable ha sido o no seleccionada. La última columna denominada *agree* cuenta el número de *TRUEs* por fila (coincidencias entre algoritmos FS).
- `plot_fs_time`, indicado por el círculo lila "9" en la figura 2.5, devuelve un gráfico de barras con el tiempo de cómputo empleado por cada algoritmo.

Clase `ml_cls`

Finalmente, el objeto `ml_fs` se emplea con la función `ml_cls`, para realizar la clasificación, validación y evaluación con datos de test (si los datos fueron previamente divididos). Esta función devuelve un objeto `ml_cls`, con multitud de posibilidades para evaluar los resultados obtenidos.

La función permite una gran variedad de ajustes, pudiéndose elegir qué métodos emplear tanto de FS, como de clasificación, así como los parámetros para la validación del modelo.

La validación implementada es del tipo CV (*cross-validation*), pudiendo configurar el valor de los *k-folds*, además de permitir realizar *repeated CV*, seleccionando el número de repeticiones que se desean emplear. También se puede seleccionar entre *Accuracy* (precisión) y el estadístico *Kappa* como medidas de rendimiento para elegir el mejor modelo. Tanto para el entrenamiento, como para la validación, se han empleado las faciliades que en este sentido aprota el paquete `caret` [9].

Los métodos que incorpora la clase *ml_cls* son muy numerosos, los más importantes de los cuales son:

- `summary`, indicado por el círculo gris "4c" en la figura 2.5, proporciona el resumen en texto específico del objeto `ml_cls`.
- `get_training_results` y `get_test_results`, indicados por el círculo gris "10" en la figura 2.5, devuelven los resultados, tanto de *Accuracy* como de *Kappa*, del proceso de CV en el primer caso y del proceso de evaluación mediante el set de test, en el segundo.
- `get_best_model` y `get_best_model_test`, indicados por el círculo gris "11" en la figura 2.5, devuelven el mejor modelo obtenido en la etapa de CV y en la evaluación de datos de test, respectivamente.
- `get_confusionMatrix_test`, indicado por el círculo verde "12" en la figura 2.5, es solo aplicable a objetos que hayan usado un set de datos previamente dividido con la función `split_data` y que hayan realizado la evaluación con datos de test. Para estos objetos, esta función devuelve la matriz de confusión para un par de combinaciones FS + clasificador, indicado por el usuario.
- `plot`, indicado por el círculo verde "13" en la figura 2.5, permite generar multitud de gráficos para ilustrar el rendimiento en cuanto a *Accuracy* y *Kappa*. Del mismo modo, también permite generar curvas ROC (*receiver operating characteristic curv*, sólo para problemas con 2 clases).
- `plot_test`, indicado por el círculo lila "14" en la figura 2.5, genera gráficos de *Accuracy* y *Kappa* para la evaluación con datos de test.
- `plot_fs_time`, indicado por el círculo lila "15" en la figura 2.5, devuelve un gráfico de barras con el tiempo de cómputo empleado por cada algoritmo, tanto en la fase de clasificación, como en la suma de FS + clasificación.

Finalmente, destacar que todos los datos generados pueden ser guardados mediante objetos de *R*, para ser usados posteriormente u en otros equipos.

Algoritmos de FS implementados

Los algoritmos de FS implementados en el paquete pueden ser consultados en cada momento mediante la función `available_fs_methods` del mismo paquete. Estos algoritmos son[1]:

- *Filter* methods:
  - random_fs
  - anova_fs
  - limma_fs
  - correlation_based_fs:
  - information_gain_fs:
  - relief_fs:
  - mRMR_fs:
- *Wrapper* methods:
  - ga_fs:
  - mseq_fs:
  - boruta_fs:

La mayoría ya han sido comentados en la introducción (ver sección 2.1.4). Sin embargo, los siguientes fueron implementados específicamente para el presente proyecto:

1. **random_fs:** símplemente selecciona el número de variables deseadas al azar.
2. **anova_fs**: basa la selección de marcadores en la significación en el test estadístico ANOVA, comparando las clases de interés.
3. **limma_fs**: método muy similar al anterior, pero en lugar de emplear un test ANOVA, realiza un ajuste de un modelo lineal con moderación del error estándar por *empirical Bayes*, tal cual está implementado en el paquete `limma` [8].
4. **ga_fs**: es un algoritmo genético de corte clásico pero adaptado para funcionar acoplado a tres clasificadores. El algoritmo empieza generando un número de conjuntos de variables del tamaño especificado que debe tener la selección final. Estas se emplean para entrenar 3 clasificadores: kNN, C5.0 (sin emplear

[1] el añadido "_fs" corresponde a *feature selection*

boosting) y LDA. Cada uno de estos clasificadores es distinto en la manera en que clasifica los datos, con lo que se reduce el sobreajuste del modelo a un tipo de clasificador. Al mismo tiempo, estos algoritmos son bastante eficientes computacionalmente. Los resultados de los tres clasificadores se promedia y, cada generación (iteración) sólo el 10 % de los mejores conjuntos se seleccionan, el resto de variables se vuelve a colocar con las no empleadas en la presente generación. Posteriormente tiene lugar la etapa de recombinación aleatoria, que sólo se da en la selección de los mejores. Seguidamente se produce la fase de mutación, que intercambia un número aleatorio (y bajo) de variables entre la población de conjuntos de variables seleccionadas y el resto de variables. En la siguiente generación, nuevos subconjuntos de variables se generan y se colocan junto con los sleccionados de la anterior generación. El algoritmo se detiene cuando llega al número de generaciones indicadas.

5. **mseq_fs**: es una versión modificada del algoritmo secuencial para la selección de variables acoplado a la misma función de *fitness* que el algoritmo genético: la precisión promedio en los clasificadores kNN, C5.0 y LDA. El algoritmo empieza seleccionando las variables que tienen un rendimiento medio de la clasificación mayor al esperado por azar (50 % para problemas de 2 clases, 33 % para problemas de 3 clases, etc). Seguidamente, cada variable se empareja con cada una de las restantes y se evalúa su precisión media en la clasificación. Sólo la o las mejores parejas se seleccionan en cada iteración, probando los subconjuntos de variables seleccionados con todas las otras variables, una a una, con la única restricción de que una misma variable no puede estar repetida en un subconjunto. El algoritmo se detiene si el añadir cualquiera de las otras variables no mejora el rendimiento de clasificación, o si el número máximo de variables a seleccionar ha sido alcanzado.

Algoritmos de clasificación implementados

Los algoritmos de clasificación implementados en el paquete pueden ser consultados en cada momento mediante la función `available_cls_methods` del mismo paquete. Estos algoritmos son (el añadido "_cls" corresponde a clasificación):

Todos ellos han sido introducidos en la introducción (ver sección 2.1.5).

- *k*-Nearest Neighbor (kNN), identificado como "knn".
- Red neuronal artificial (ANN), identificado como "nnet".
- Máquinas de soporte vectorial (SVM). Estas se han implementado con *kernel* lineal (identificado como "SVMLinear") y con *kernel* Gusiano (identificado como "SVMRadial").

- Árboles de decisión, empleando el algoritmo C5.0, identificado como “C5.0”.
- Random Forest, identificado como “rf”.
- Glmnet, identificado como “glmnet”.
- Linear Discriminant Analysis, identificado como “lda”.

2.2.2. La aplicación *Shiny* de *methylearning*

Para facilitar el acceso a las funcionalidades que aporta el paquete `methylearning` se ha diseñado una aplicación web empleando el paquete *Shiny*, de `R`. Esta aplicación funciona como una interfaz gráfica para los análisis que se pueden llevar a cabo con el paquete `methylearning`. Por lo tanto, está también estructurada para ser usada de forma secuencial, a partir de tres paneles accesibles por un sistema de pestañas: “DATA LOADING”, “FEATURE SELECTION” y “CLASSIFICATION”.

Al ejecutar la aplicación (ver las instrucciones en el archivo “README.md” incluido con el software) lo primero que se ha incorporado es un mensaje de bienvenida que explica resumidamente el funcionamiento de la aplicación (ver figura 2.6).

Después de pulsar *START* en el mensaje de bienvenida, para que desaparezca, aparece el primer panel identificado con la pestaña “DATA LOADING”. Este panel es el encargado de la carga de los datos para el análisis. Incorpora las dos posibles fuentes de entrada que ya incorpora el paquete *methylearning*: un set de datos propio en forma de un archivo con extensión `*.csv` o un código de acceso de la base de datos GEO y el nombre de la columna a usar como etiquetas de clase. Además, con fines demostrativos, se ha incluido un pequeño set de datos, el mismo que se analizará en la sección 2.3 de esta memoria. Además, se incorpora una pequeña pestaña para introducir la semilla de aleatoriedad que se usará para generar resultados reproducibles. Tras la confirmación (pulsando “CONFIRM”), en breves instantes aparece el sumario de los datos cargados y una pequeña muestra del set de datos (ver figura 2.7).

Una vez los datos han sido cargados, se procede a acceder a la pestaña “FEATURE SELECTION”, para llevar a cabo la selección de marcadores. Se puede seleccionar uno o más algoritmos de FS, a la vez que se debe indicar el número máximo de marcadores (variables) a seleccionar por cada algoritmo, así como el número de *central processor units* (CPUs) a emplear en las partes paralelizadas de la herramienta. Tras pulsar el botón “PERFORM FS” aparece un mensaje indicando que los cálculos se están realizando (pueden tardar bastante dependiendo de la dimensionalidad de los datos de entrada y de los algoritmos escogidos). Al completar la

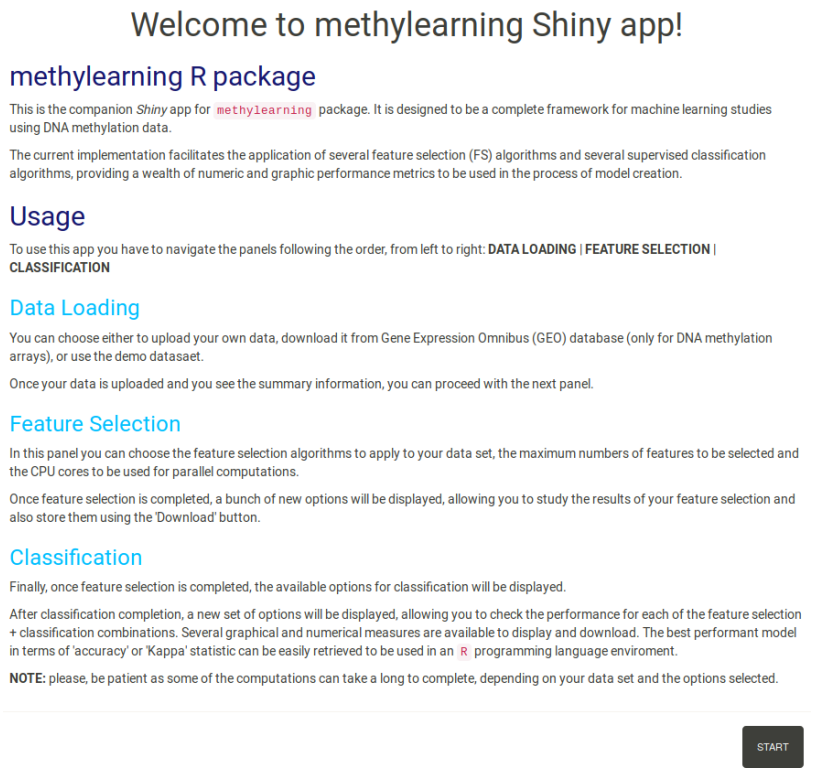

Figura 2.6: Mensaje de bienvenida de la aplicación web.

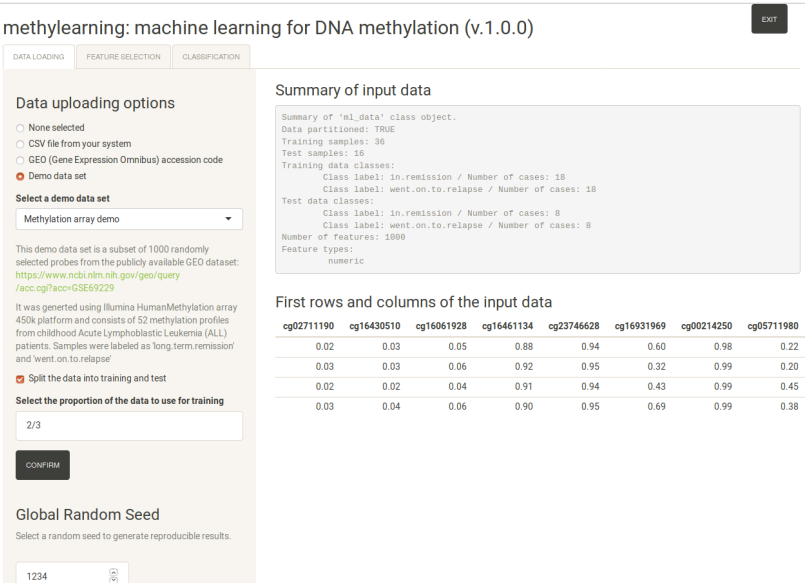

Figura 2.7: Primera parte del análisis: carga de los datos.

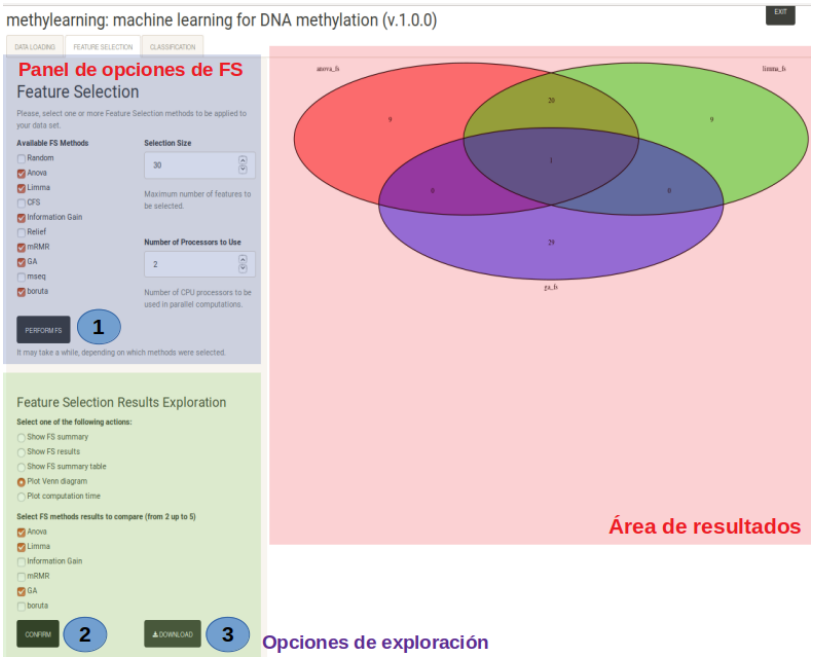

Figura 2.8: Exploración y descarga de los resultados de la FS. En azul se resalta el área de opciones para la FS, junto con el botón "PERFORM FS" que inicia la FS (1). En verde se indica el panel de opciones secundario que se despliega al acabar la FS y que permite explorar los resultados (2) o descargarlos (3). En rojo se muestra el área donde aparecen los resultados a explorar, seleccionados en el panel de opciones de exploración.

FS, se despliega un nuevo panel de opciones, debajo del anterior, donde se pueden explorar de múltiples maneras y descargar los resultados obtenidos (ver figura 2.8).

Finalmente, en la tercera pestaña de la aplicación ("CLASSIFICATION") aparecen una serie de opciones, dependiendo de los algoritmos escogidos en la sección anterior. Aquí, se puede elegir qué combinaciones de FS y algoritmos de clasificación probar, los parámetros de la validación cruzada, si se realiza o no la evaluación de los resultados mediante la partición de "test" (si esta fué creada en la primera pestaña) y el número de CPUs a emplear para los cálculos paralelizables. Tras seleccionar todas las opciones, se pulsa el botón "PERFORM CLS" para empezar la clasificación. Este proceso también puede tomar bastante tiempo en completarse, momento en el que el mensaje con la barra de progreso desaparece y se despliega un nuevo menú con las opciones de exploración de los resultados y de descarga de los mismos (ver figura 2.9). Vale la pena destacar que en este último menú desplegable se destaca bien visible un botón que permite la descarga de un objeto de `R` que contiene una lista con el objeto de la clase `caret::train` con el mejor modelo seleccionado, ya entrenado para ser usado en cualquier otra sesión de `R`, así como

Figura 2.9: Exploración y descarga de los resultados de la clasificación. En azul se resalta el área de opciones para la clasificación y validación, junto con el botón "PERFORM CLASSIFICATION" para iniciar la clasificación (1). En verde se indica el panel de opciones secundario que se despliega al acabar la clasificación y validación. Para confirmar las diferentes opciones se usa el botón "CONFIRM" (2) y el botón "DOWNLOAD DISPLAYED RESULTS" (3) para la descarga de los resultados visualizados. Se ha añadido un último botón (4) que permite descargar el mejor modelo en forma de objeto de `R`.

información adicional.

Una de las constantes que ha condicionado el diseño de toda la aplicación es el tiempo necesario para completar algunos cálculos. Por esta razón, se ha optado por eliminar la mayoría de la reactividad de la que hace gala *Shiny*, mediante la incorporación de botones para confirmar las selecciones y los cambios. De este modo, se mejora la usabilidad de la herramienta y se reducen la cantidad de cálculos a realizar.

2.3. Análisis de demostración: clasificación de pacientes de ALL según su pronóstico

En la presente sección se mostrará un análisis completo realizado con el paquete `methylearning`, a modo de demostración de sus capacidades. Se empleará directamente el paquete, accediendo a sus clases y funciones, desde una sesión de `R`. Sin embargo, el mismo análisis puede ser realizado mediante la aplicación *Shiny*

asociada. Este análisis está basado en el que se puede encontrar en la *vignette* que acompaña al paquete `methylearning`.

2.3.1. Preparación de los datos

Los datos de partida para el presente análisis están disponibles públicamente en GEO, con el siguiente código de acceso: GSE69229. Estos datos corresponden al trabajo de Alem S Gabriel et al., publicado en la revista *Epigenetics* en 2015 [58]. El objetivo de ese trabajo era utilizar los patrones de metilación para la búsqueda de biomarcadores que pudiesen predecir la recaída que se produce en algunos pacientes con leucemia linfoblástica aguda (*acute lymphoblastic leukemia*, ALL) infantil. La ALL infantil generalmente tiene un buen pronóstico y los pacientes responden bien al tratamiento. No obstante, una parte de ellos recaen, habiendo una baja supervivencia tras las recaídas. Por tanto, la búsqueda de biomarcadores que determinen si un paciente recaerá o no después del tratamiento es de vital importancia. Con este fin, se empleó el *Illumina HumanMethylation450 BeadChip* (ver sección 2.1.2) para determinar el patrón de metilación de 52 pacientes, a partir de muestras obtenidas en el momento del diagnóstico (antes del tratamiento). De estos 52 pacientes, la mitad (26) sufrieron una recaída, mientras que la otra mitad lograron una remisión de largo plazo.

De este modo, el set de datos disponible contiene los valores de metilación (β *values*) para unos 450000 sitios CpGs en estas 52 muestras. Para la descarga de estos datos y la carga en una sesión interactiva de `R` se hará uso de la función `GEOquery::getGEO`:

```
# Carga de la librería methylearning.
library(methylearning)
# Carga de las librerías necesarias para la descarga.
library(GEOquery)
library(Biobase)

# Descarga de los datos de la base de datos de GEO.
gse <- getGEO("GSE69229", getGPL = FALSE)
```

Una vez se dispone de los datos descargados en forma de objeto `GEOquery::GSE`, este se puede usar para crear un objeto `methylearning::ml_data`, punto de partida del análisis, utilizando la función `methylearning::get_GEO_methylarray`. Debido al tamaño de los datos con unas 450000 CpGs x 52 muestras, para este análisis de demostración se ha optado por reducir el set de datos a 1000 CpGs escogidas de manera aleatoria. Para ello, se puede emplear la opción de reducir los datos mediante un filtro aleatorio (`random_filter`). Es importante destacar que previamen-

te se ha tenido que explorar el objeto `GEOquery::GSE` para determinar el nombre de la columna que actuará como etiquetas de clase, en este caso llamada "outcome:ch1", contenida en el `AnnotatedDataFrame` "phenoData", contenido en el objeto `GEOquery::GSE`.

```
# Create an ml_data object.
set.seed(1234)
ml <- get_GEO_methylarray(gse, target_name = "outcome:ch1",
                          random_filter = 1000)
summary(ml)
```

Para agilizar el proceso de carga de datos, en la distribución del paquete `methylearning` se adjunta una tabla con 1000 CpGs seleccionadas al azar para las 52 muestras de este set de datos, en forma de `data.frame`, listo para ser usado con la función `methylearning::ml_data`.

```
demo <- "/path/to/methylearning/vignettes/demo_data/demo_1.RData"
load(demo)
# Se carga un data.frame de nombre 'df', en el que la columna con las
# etiquetas de clase es la útlima.
ml <- ml_data(df, labels_column = dim(df)[2])
summary(ml)
```

Al ejecutar el código anterior se genera el sumario del objeto `ml` de la clase `ml_data`

```
Summary of 'ml_data' class object.
Data partitioned: FALSE
Training samples: 52
Test data not present.
Training data classes:
    Class label: in.remission / Number of cases: 26
    Class label: went.on.to.relapse / Number of cases: 26
Number of features: 1000
Feature types:
     numeric
```

Una vez los datos han sido cargados, por defecto el objeto contiene todos los datos como datos de entrenamiento. Para generar particiones de los datos se puede emplear la función `methylearning::split_data`, que genera particiones de los datos en la proporción especificada, manteniendo en lo posible la misma proporción de clases en los datos de test y de entrenamiento, ya que internamente se usa

`caret::craeteDataPartition`. La partición de los datos es sólo recomendable para sets de datos con muchas muestras. En el presente caso, no sería muy recomendable, ya que el total de muestras es de tan solo 52. De todos modos, se ha procedido a particionar los datos, para mostrar las funcionalidades adicionales para datos particionados.

```
# Se dividirán los datos en 2/3 para entrenamiento y 1/3 para test.
set.seed(1234)
ml_s <- split_data(ml, splitting = 2/3)
summary(ml_s)
```

```
Summary of 'ml_data' class object.
Data partitioned: TRUE
Training samples: 36
Test samples: 16
Training data classes:
    Class label: in.remission / Number of cases: 18
    Class label: went.on.to.relapse / Number of cases: 18
Test data classes:
    Class label: in.remission / Number of cases: 8
    Class label: went.on.to.relapse / Number of cases: 8
Number of features: 1000
Feature types:
     numeric
```

La última llamada a `summary` demuestra que los datos han sido particionados correctamente.

2.3.2. *Feature selection*

Realizar la selección de variables empleando diversos métodos es una tarea muy sencilla, empelando la función `methylearning::ml_fs`, que acepta objetos de la clase `methylearning::ml_data`.

Para conocer el identificador de todos los métodos de FS implementados en el paquete, se ha incorporado la función `methylearning::available_fs_methods`:

```
available_fs_methods(verbose = TRUE)
```

```


Available Feature Selection Methods.
Filter methods:
```

```
    .random_fs: Random Feature Selection. For testing purposes.
    .anova_fs: ANOVA feature selection.
    .limma_fs: Select CpGs by differential methylation using limma.
   Similar to
               anova_fs but optimized for methylation arrays. Do not use
   with
               bisulfite seq.
    .correlation_based_fs: Correlation-based Feature Selection. From
   FSelector
                           package.
    .information_gain_fs: Entropy-based information gain feature
   selection. From
               FSelector package.
    .relief_fs: RELIEF algorithm for feature selection. From FSelector.
    .mRMR_fs: Minimum redundancy, maximum relevance feature filtering.
   Classic
              version from mRMRe package. Due to implementation details,
   the
              number of features should be <= 46340.
Wrapper methods:
    .ga_fs: Wrapped Genetic Algorithm - knn/C5.0/lda for feature
   selection.
    .mseq_fs: Modified Sequential Feature Selection.
    .boruta_fs: Boruta Algorithm for Feature Selection. From Boruta
   package.
```

El argumento `verbose = TRUE` produce que se imprima la lista acompañada de una pequeña explicación para cada método.

Para seleccionar los algoritmos de FS que se van a emplear, estos se deben indicar en forma de `character vector` en el argumento `fs_methods` de la función `ml_fs`. Opcionalmente, se puede indicar simplemente "all", si se desean usar todos, "filters" si se desean usar todos los `filters` o "wrappers" si se desean usar todos los `wrappers`.

Muchos de los algoritmos implementados exigen que se les indique el tamaño de la selección (número de variables a seleccionar). Otros no lo exigen, pero pueden devolver un número muy grande y poco práctico. Finalmente, algunos reportan muy pocas variables, a pesar de lo grande del set de datos. Para homogeneizar los resultados, se ha implementado la posibilitar de ajustar el límite máximo de variables a seleccionar, mediante el argumento `selection_size`. No se ha implementado un límite mínimo.

```
set.seed(1234)
```

```
fs_methods_to_use <- c("random_fs", "anova_fs", "limma_fs",
                       "correlation_based_fs", "information_gain_fs",
                       "relief_fs", "mRMR_fs", "ga_fs",
                       "boruta_fs")
ml_f <- ml_fs(ml_s, fs_methods = fs_methods_to_use, selection_size = 20,
              cores = 4)
```

Es necesario recordar que algunos de los métodos implementados son muy costosos computacionalmente y pueden tardar bastante en completarse, además de requerir grandes cantidades de memoria RAM. Se recomienda usar la paralelización implementada, mediante el parámetro `cores`, que indica el número de núcleos de CPU a usar. Precisamente, esta es la razón de que el método "mseq_fs", la implementación propia del método secuencial de selección de marcadores, no se haya ejecutado. Este algoritmo es muy exhaustivo y tremendamente costoso a nivel de tiempo de computación y memoria del sistema, con lo que requiere de una máquina muy potente, incluso para un set de datos reducido de tan solo 1000 variables.

Como resultado de la FS se genera un objeto de la clase `methylearning::ml_fs`, que hereda de la clase `methylearning::ml_fs`, por lo que conserva sus atributos y métodos.

Por defecto, la selección de variables se hace a partir del conjunto total de datos, independientemente de si estos se particionaron o no. Este comportamiento se puede cambiar, estableciendo el parámetro `use_all_data = FALSE` de la función `methylearning::ml_fs`, para que sólo se tengan en cuenta los datos de entrenamiento y mantener los dos sets de datos completamente independientes en todo el proceso.

Los resultados obtenidos se pueden explorar fácilmente utilizando los métodos implementados. Por ejemplo, el método `summary` devuelve los resultados más relevantes de la FS.

```
summary(ml_f)
```

```
Selection methods applied:
     random_fs, anova_fs, limma_fs, correlation_based_fs,
   information_gain_fs, relief_fs, mRMR_fs, ga_fs, boruta_fs

Number of features selected:
    random_fs: 20 features selected
    anova_fs: 20 features selected
    limma_fs: 20 features selected
    correlation_based_fs: 5 features selected
    information_gain_fs: 5 features selected
```

```
    relief_fs: 20 features selected
    mRMR_fs: 20 features selected
    ga_fs: 20 features selected
    boruta_fs: 2 features selected

Computation time for each feature selection method:
    anova_fs: 0.777 seconds
    mRMR_fs: 0.663 seconds
    ga_fs: 163.219 seconds
    boruta_fs: 5.59 seconds
    random_fs: 0 seconds
    limma_fs: 0.192 seconds
    correlation_based_fs: 172.666 seconds
    information_gain_fs: 1.982 seconds
    relief_fs: 521.618 seconds

Summary of 'ml_data' class object.
Data partitioned: TRUE
Training samples: 36
Test samples: 16
Training data classes:
    Class label: in.remission / Number of cases: 18
    Class label: went.on.to.relapse / Number of cases: 18
Test data classes:
    Class label: in.remission / Number of cases: 8
    Class label: went.on.to.relapse / Number of cases: 8
Number of features: 1000
Feature types:
     numeric
```

Para obtener las listas de variables seleccionadas por cada algoritmo se puede emplear la función `methylearning::selection_results`:

```
selection_results(ml_f)
```

```
$anova_fs
 [1] "cg18003698" "cg15722265" "cg23420331" "cg18987026" "cg25918510"
 [6] "cg20846148" "cg20575848" "cg25708695" "cg11313124" "cg04258333"
[11] "cg00112048" "cg11343579" "cg02625024" "cg05822031" "cg04772797"
[16] "cg27286337" "cg16105754" "cg06104975" "cg09965939" "cg13403615"

$boruta_fs
```

```
[1] "cg23420331" "cg04772797"

$correlation_based_fs
[1] "cg04772797" "cg23420331" "cg04974121" "cg01250320" "cg04258333"

$ga_fs
 [1] "cg04826831" "cg08381325" "cg19230787" "cg04015912" "cg05735590"
 [6] "cg15722265" "cg20167762" "cg05770971" "cg25633067" "cg23795217"
[11] "cg26169668" "cg01148088" "cg06400318" "cg06604811" "cg01846046"
[16] "cg01154537" "cg05079191" "cg25498327" "cg02625024" "cg04674081"

$information_gain_fs
[1] "cg23420331" "cg04772797" "cg04974121" "cg01250320" "cg04258333"

$limma_fs
 [1] "cg18003698" "cg15722265" "cg23420331" "cg05822031" "cg25506432"
 [6] "cg11343579" "cg11313124" "cg26394916" "cg13403615" "cg11125805"
[11] "cg00584602" "cg04772797" "cg00112048" "cg02625024" "cg16105754"
[16] "cg05735590" "cg01056135" "cg20575848" "cg04258333" "cg16734526"

$mRMR_fs
 [1] "cg13649552"     "cg05770971"     "cg19106945"     "ch.4.69646068F"
 [5] "cg25315312"     "cg14585793"     "cg04895363"     "cg16875863"
 [9] "cg18306943"     "cg12966915"     "cg22329153"     "cg14954466"
[13] "cg15576859"     "cg08656816"     "cg16593081"     "cg05602987"
[17] "cg06755438"     "cg13063101"     "cg02128308"     "cg27068619"

$random_fs
 [1] "cg05822031"     "cg22825487"     "cg26394916"     "cg24438217"
 [5] "cg08876470"     "cg16566353"     "cg16211055"     "cg05536984"
 [9] "cg09275000"     "cg01056135"     "cg07945148"     "cg04231427"
[13] "cg17474545"     "cg07089235"     "cg06770536"     "cg22057372"
[17] "cg19734433"     "ch.9.98937537R" "cg10390329"     "cg21953346"

$relief_fs
 [1] "cg08994891" "cg06209035" "cg14603227" "cg22969661" "cg05585303"
 [6] "cg19797013" "cg00366603" "cg17354708" "cg02900356" "cg01751245"
[11] "cg27145044" "cg23420331" "cg09827249" "cg10754596" "cg21786334"
[16] "cg06308972" "cg07717774" "cg13133420" "cg10401803" "cg25708695"
```

También, mediante la función `methylearning::selection_df` se puede obtener un `data.frame` en el que comprobar el grado de coincidencia de cada algoritmo y,

mediante la columna “agree”, se puede saber el número de algoritmos que han seleccionado cada una de las variables.

```
head(selection_df(ml_f))
```

```
           anova_fs boruta_fs correlation_based_fs
cg23420331     TRUE      TRUE                 TRUE
cg04772797     TRUE      TRUE                 TRUE
cg04258333     TRUE     FALSE                 TRUE
cg15722265     TRUE     FALSE                FALSE
cg02625024     TRUE     FALSE                FALSE
cg05822031     TRUE     FALSE                FALSE

           ga_fs information_gain_fs limma_fs
cg23420331 FALSE                TRUE     TRUE
cg04772797 FALSE                TRUE     TRUE
cg04258333 FALSE                TRUE     TRUE
cg15722265  TRUE               FALSE     TRUE
cg02625024  TRUE               FALSE     TRUE
cg05822031 FALSE               FALSE     TRUE

           mRMR_fs random_fs relief_fs agree
cg23420331   FALSE     FALSE      TRUE     6
cg04772797   FALSE     FALSE     FALSE     5
cg04258333   FALSE     FALSE     FALSE     4
cg15722265   FALSE     FALSE     FALSE     3
cg02625024   FALSE     FALSE     FALSE     3
cg05822031   FALSE      TRUE     FALSE     3
```

Se pueden obtener dos tipos de gráficos a partir de los objetos de clase `ml_fs`, un diagrama de Venn, que mostrará el solapamiento de entre 2 y 5 algoritmos de FS, y un diagrama de barras con los tiempos de computación de cada algoritmo.

```
# Diagrama de Venn de la selección de ANOVA vs la de limma.
meths <- c("anova_fs", "limma_fs")
fill <- c("red", "blue")
plot_venn_fs(ml_f, fs_methods = meths, fill = fill, category = meths)
```

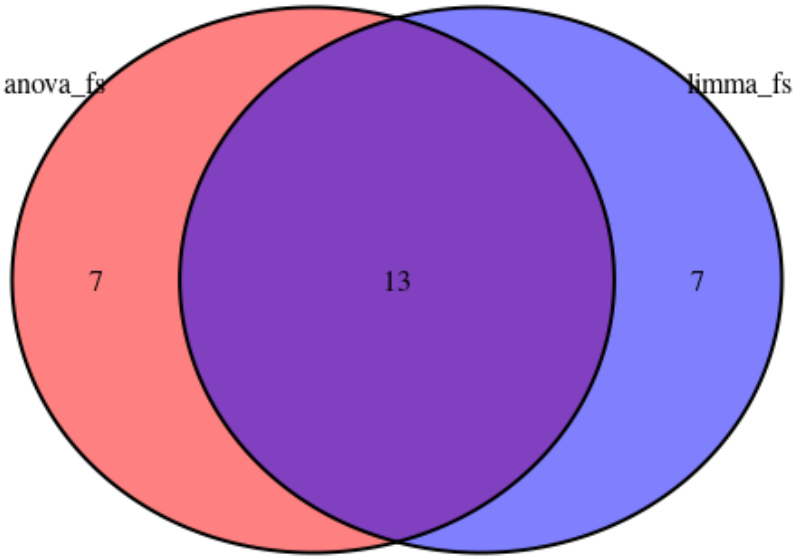

Figura 2.10

```
# Diagrama de Venn de la selección de ANOVA vs limma vs
# correlation_based_fs vs information_gain_fs.
meths <- c("anova_fs", "limma_fs", "correlation_based_fs",
      "information_gain_fs")
fill <- c("red", "blue", "orange", "green")
plot_venn_fs(ml_f, fs_methods = meths, fill = fill, category = meths)
```

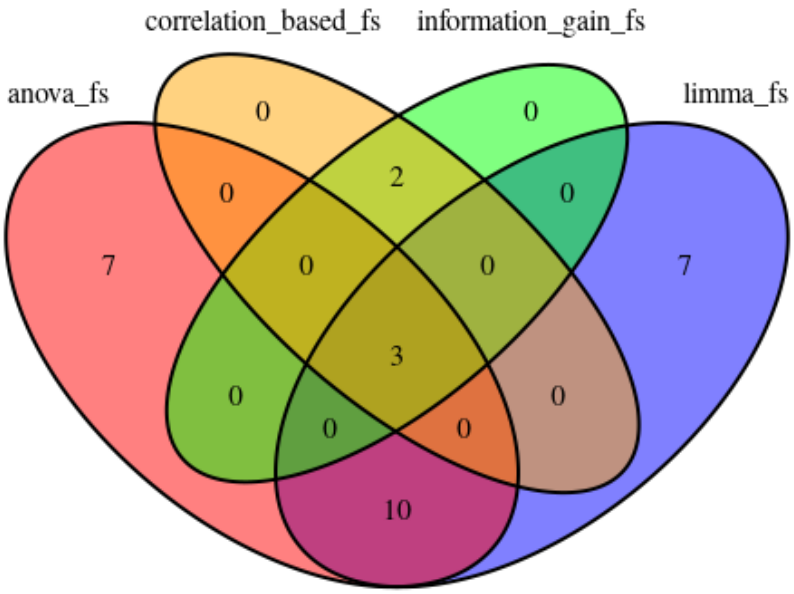

Figura 2.11

Se puede ver como las selecciones por el método “ANOVA” y “limma” son muy similares. Además, las selecciones por los métodos “correlation_based_fs” e “information_gain_fs” son idénticas.

```
# Diagrama de barras con los tiempos de computación.
plot_fs_time(ml_f)
```

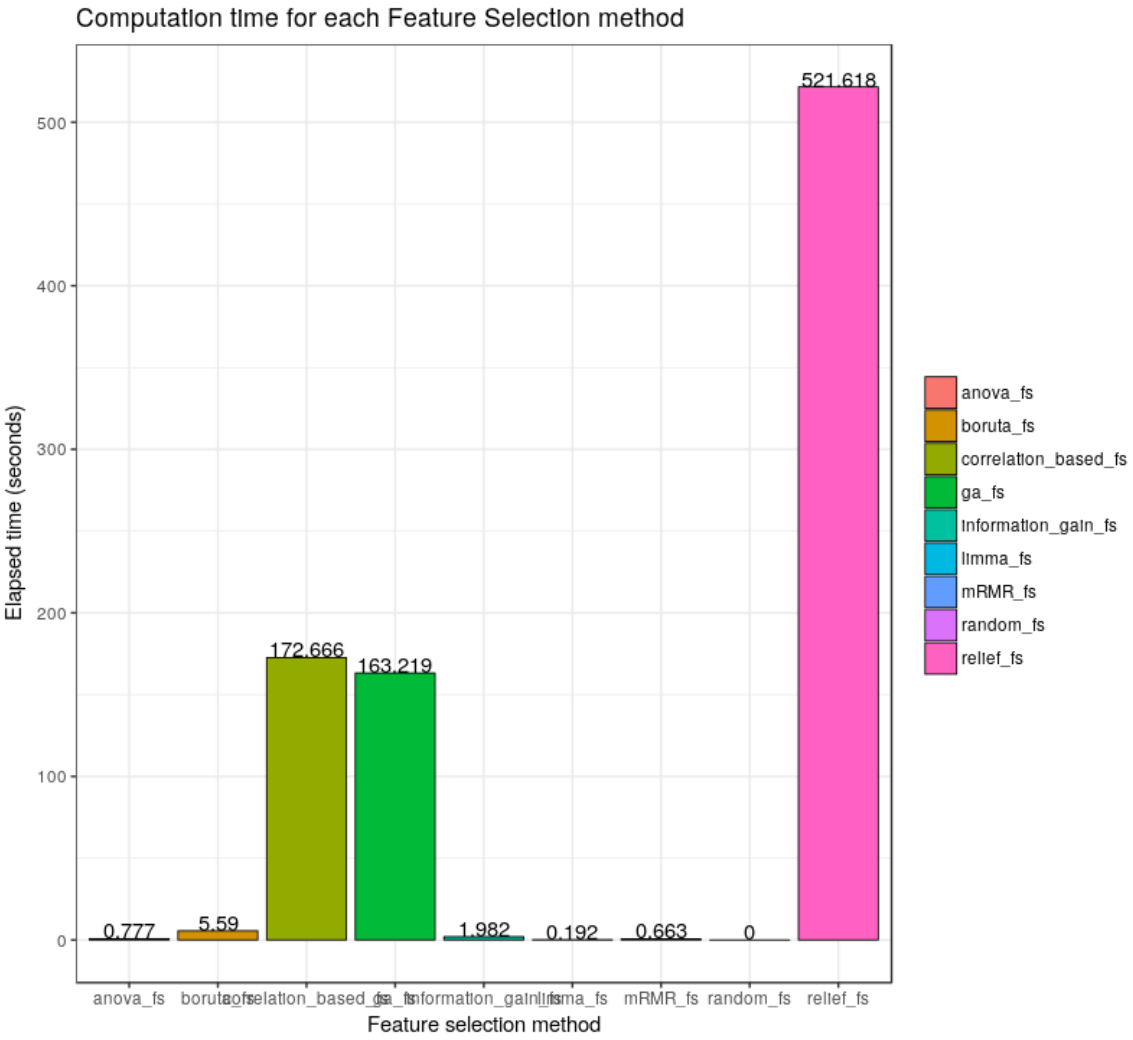

Figura 2.12

Al analizar este resultado se puede ver como hay tres métodos que son mucho más lentos que los demás, siendo dos de ellos *filters*, que se suponen más rápidos que los *wrappers*. Es muy destacable que el algoritmo “correlation based FS” sea muchísimo más lento que “information gain FS”, a pesar de que llegan a exactamente en mismo set de variables seleccionadas. También es interesante hacer notar que el algoritmo “Boruta”, a pesar de ser un *wrapper*, es bastante rápido de ejecutar.

2.3.3. Clasificación y validación de los modelos

A partir del objeto de la clase `methylearning::ml_fs`, que contiene tanto los datos como las diferentes selecciones de variables, se pueden aplicar los métodos de clasificación fácilmente mediante la función `methylearning::ml_cls`.

Para conocer los métodos de clasificación disponibles se puede usar la función `methylearning::available_cls_methods`.

```
available_cls_methods()
```

```
[1] "knn"       "C5.0"      "rf"        "svmLinear" "svmRadial" "nnet"
    "glmnet"
[8] "lda"
```

La función `methylearning::ml_cls`, por defecto, prueba todos los métodos de clasificación implementados sobre todos los conjuntos de variables seleccionados, disponibles en el objeto de la clase `methylearning::ml_fs`. Para la validación, por defecto se realiza 3x 10-*fold cross-validation*, probando 10 posibilidades para los parámetros de cada modelo y seleccionando el mejor en base al estadístico *Kappa*. Todo esto puede ser ajustado mediante los argumentos de la función `methylearning::ml_cls`. Esta función también admite paralelización para mejorar el rendimiento, ajustando el valor del argumento `cores` al número de CPUs deseado.

Para este caso concreto, se usarán los conjuntos de variables más representativos, con los que se entrenarán los siguientes clasificadores: "knn", "svmRadial", "rf", "lda". El método de validación será 10-*fold cross-valiadtion*, sin repetición, probando solamente 3 parámetros para cada modelo, para agilizar los cálculos. Además, se realizará una evaluación del rendimiento del mejor modelo obtenido para cada combinación de FS + clasificador, usando los datos de test (`test_eval = TRUE`).

```
set.seed(1234)
fs_methods_to_use <- c("random_fs", "limma_fs",
                       "information_gain_fs",
                       "relief_fs", "mRMR_fs", "ga_fs",
                       "boruta_fs")
cls_methods_to_use <- c("knn", "svmRadial", "rf", "lda")
ml_c <- ml_cls(ml_f,
               fs_methods = fs_methods_to_use,
               cls_methods = cls_methods_to_use,
               cv_folds = 10,
               rep = 1,
```

```
              tune_length = 3,
              metric = "Kappa",
              test_eval = TRUE,
              cores = 1)
```

2.3.4. Exploración de los resultados

La exploración de los resultados obtenidos se puede hacer mediante todo un surtido de métodos implementados para la clase `methylearning::ml_cls`, que proporcionan datos tanto numéricos como gráficos.

El método `summary`, para un resumen en texto de los resultados más relevantes.

```
summary(ml_c)
```

```
Summary for object 'ml_c' of the 'ml_cls' class

Feature selection methods performed:
     random_fs, limma_fs, information_gain_fs, relief_fs, mRMR_fs, ga_fs,
    boruta_fs

Classification methods performed:
     knn, svmRadial, rf, lda

Full data set dimensions:
    Number of features: 1000
    Number of training samples: 36
    Number of test samples: 16
    Class labels: in.remission, went.on.to.relapse

Validation method used in training data classification:
     10 fold x 1 repeated cross-validation, testing 3 values for each
    tuning parameter and selecting models by Kappa

Best model selected by Kappa:
    Best feature selection method: limma_fs
    Best classification method: rf
    Best accuracy: 0.833333333333333
    Best Kappa: 0.64

Computation time:
```

```
random_fs + knn:
    Feature Selection only: 0 seconds.
    Classification only: 1.045 seconds.
    Total: 1.045 seconds.
random_fs + svmRadial:
    Feature Selection only: 0 seconds.
    Classification only: 2.514 seconds.
    Total: 2.514 seconds.
random_fs + rf:
    Feature Selection only: 0 seconds.
    Classification only: 1.312 seconds.
    Total: 1.312 seconds.
random_fs + lda:
    Feature Selection only: 0 seconds.
    Classification only: 0.49 seconds.
    Total: 0.49 seconds.
limma_fs + knn:
    Feature Selection only: 0.192 seconds.
    Classification only: 0.571 seconds.
    Total: 0.763 seconds.
limma_fs + svmRadial:
    Feature Selection only: 0.192 seconds.
    Classification only: 0.979 seconds.
    Total: 1.171 seconds.
limma_fs + rf:
    Feature Selection only: 0.192 seconds.
    Classification only: 0.906 seconds.
    Total: 1.098 seconds.
limma_fs + lda:
    Feature Selection only: 0.192 seconds.
    Classification only: 0.49 seconds.
    Total: 0.682 seconds.
information_gain_fs + knn:
    Feature Selection only: 1.982 seconds.
    Classification only: 0.547 seconds.
    Total: 2.529 seconds.
information_gain_fs + svmRadial:
    Feature Selection only: 1.982 seconds.
    Classification only: 0.887 seconds.
    Total: 2.869 seconds.
information_gain_fs + rf:
    Feature Selection only: 1.982 seconds.
```

```
    Classification only: 0.747 seconds.
    Total: 2.729 seconds.
information_gain_fs + lda:
    Feature Selection only: 1.982 seconds.
    Classification only: 0.482 seconds.
    Total: 2.464 seconds.
relief_fs + knn:
    Feature Selection only: 521.618 seconds.
    Classification only: 0.573 seconds.
    Total: 522.191 seconds.
relief_fs + svmRadial:
    Feature Selection only: 521.618 seconds.
    Classification only: 0.935 seconds.
    Total: 522.553 seconds.
relief_fs + rf:
    Feature Selection only: 521.618 seconds.
    Classification only: 0.965 seconds.
    Total: 522.583 seconds.
relief_fs + lda:
    Feature Selection only: 521.618 seconds.
    Classification only: 0.494 seconds.
    Total: 522.112 seconds.
mRMR_fs + knn:
    Feature Selection only: 0.663 seconds.
    Classification only: 0.557 seconds.
    Total: 1.22 seconds.
mRMR_fs + svmRadial:
    Feature Selection only: 0.663 seconds.
    Classification only: 0.935 seconds.
    Total: 1.598 seconds.
mRMR_fs + rf:
    Feature Selection only: 0.663 seconds.
    Classification only: 0.968 seconds.
    Total: 1.631 seconds.
mRMR_fs + lda:
    Feature Selection only: 0.663 seconds.
    Classification only: 0.488 seconds.
    Total: 1.151 seconds.
ga_fs + knn:
    Feature Selection only: 163.219 seconds.
    Classification only: 0.549 seconds.
    Total: 163.768 seconds.
```

```
    ga_fs + svmRadial:
        Feature Selection only: 163.219 seconds.
        Classification only: 0.95 seconds.
        Total: 164.169 seconds.
    ga_fs + rf:
        Feature Selection only: 163.219 seconds.
        Classification only: 0.937 seconds.
        Total: 164.156 seconds.
    ga_fs + lda:
        Feature Selection only: 163.219 seconds.
        Classification only: 0.49 seconds.
        Total: 163.709 seconds.
    boruta_fs + knn:
        Feature Selection only: 5.59 seconds.
        Classification only: 0.55 seconds.
        Total: 6.14 seconds.
    boruta_fs + svmRadial:
        Feature Selection only: 5.59 seconds.
        Classification only: 0.876 seconds.
        Total: 6.466 seconds.
    boruta_fs + rf:
        Feature Selection only: 5.59 seconds.
        Classification only: 0.553 seconds.
        Total: 6.143 seconds.
    boruta_fs + lda:
        Feature Selection only: 5.59 seconds.
        Classification only: 0.484 seconds.
        Total: 6.074 seconds.

Summary of classification results using training data:
$Accuracy
                          knn svmRadial        rf       lda
random_fs           0.4666667 0.5666667 0.6083333 0.5666667
limma_fs            0.7833333 0.7666667 0.8333333 0.6916667
information_gain_fs 0.6333333 0.5416667 0.7083333 0.6083333
relief_fs           0.4500000 0.5333333 0.5416667 0.4500000
mRMR_fs             0.5583333 0.6500000 0.4500000 0.4500000
ga_fs               0.6500000 0.4916667 0.6833333 0.6833333
boruta_fs           0.6583333 0.7833333 0.7833333 0.6500000

$Kappa
                       knn svmRadial    rf   lda
```

```
random_fs           −0.08      0.00  0.15  0.14
limma_fs             0.58      0.51  0.64  0.39
information_gain_fs  0.25      0.10  0.45  0.20
relief_fs           −0.08      0.00  0.09 −0.10
mRMR_fs              0.14      0.20 −0.09 −0.15
ga_fs                0.34      0.01  0.34  0.35
boruta_fs            0.33      0.54  0.58  0.32

Data partitioned: TRUE
Summary of classification results using test data:
$Accuracy
                       knn svmRadial     rf    lda
random_fs           0.6875    0.3750 0.4375 0.5000
limma_fs            0.7500    0.7500 0.7500 0.8125
information_gain_fs 0.8125    0.7500 0.5625 0.5000
relief_fs           0.5000    0.1875 0.6875 0.5625
mRMR_fs             0.6250    0.4375 0.5000 0.6250
ga_fs               0.4375    0.5625 0.6250 0.7500
boruta_fs           0.9375    0.8125 0.6250 0.7500

$Kappa
                       knn svmRadial     rf   lda
random_fs            0.375    −0.250 −0.125 0.000
limma_fs             0.500     0.500  0.500 0.625
information_gain_fs  0.625     0.500  0.125 0.000
relief_fs            0.000    −0.625  0.375 0.125
mRMR_fs              0.250    −0.125  0.000 0.250
ga_fs               −0.125     0.125  0.250 0.500
boruta_fs            0.875     0.625  0.250 0.500
```

Los resultados tanto para *Accuracy* como para el estadístico *Kappa* se pueden obtener fácilmente usando `get_training_results` para los resultados de la validación del set de datos de entrenamiento y `get_test_results` para los resultados de las predicciones con los datos de test.

```
# Resultados de los datos de training.
get_training_results(ml_c)
```

```
$Accuracy
                         knn svmRadial        rf       lda
random_fs           0.4666667 0.5666667 0.6083333 0.5666667
limma_fs            0.7833333 0.7666667 0.8333333 0.6916667
```

```
information_gain_fs 0.6333333 0.5416667 0.7083333 0.6083333
relief_fs           0.4500000 0.5333333 0.5416667 0.4500000
mRMR_fs             0.5583333 0.6500000 0.4500000 0.4500000
ga_fs               0.6500000 0.4916667 0.6833333 0.6833333
boruta_fs           0.6583333 0.7833333 0.7833333 0.6500000

$Kappa
                       knn svmRadial    rf   lda
random_fs           −0.08      0.00  0.15  0.14
limma_fs             0.58      0.51  0.64  0.39
information_gain_fs  0.25      0.10  0.45  0.20
relief_fs           −0.08      0.00  0.09 −0.10
mRMR_fs              0.14      0.20 −0.09 −0.15
ga_fs                0.34      0.01  0.34  0.35
boruta_fs            0.33      0.54  0.58  0.32
```

```
# Resultados de los datos de test.
get_test_results(ml_c)
```

```
$Accuracy
                       knn svmRadial     rf    lda
random_fs           0.6875    0.3750 0.4375 0.5000
limma_fs            0.7500    0.7500 0.7500 0.8125
information_gain_fs 0.8125    0.7500 0.5625 0.5000
relief_fs           0.5000    0.1875 0.6875 0.5625
mRMR_fs             0.6250    0.4375 0.5000 0.6250
ga_fs               0.4375    0.5625 0.6250 0.7500
boruta_fs           0.9375    0.8125 0.6250 0.7500

$Kappa
                       knn svmRadial     rf   lda
random_fs            0.375    −0.250 −0.125 0.000
limma_fs             0.500     0.500  0.500 0.625
information_gain_fs  0.625     0.500  0.125 0.000
relief_fs            0.000    −0.625  0.375 0.125
mRMR_fs              0.250    −0.125  0.000 0.250
ga_fs               −0.125     0.125  0.250 0.500
boruta_fs            0.875     0.625  0.250 0.500
```

Viendo estos resultados, sorprende el hecho de que el modelo con el *Kappa* más alto para los datos de entrenamiento sea la combinación limma + rf. Sin embargo,

los datos de test han arrojado resultados diferentes, siendo la mejor combinación Boruta + kNN. Esto, seguramente es debido al tamaño tan reducido del set de datos, problema agravado por la partición de los mismos, lo que provoca que ninguna de las dos validaciones (CV con los datos de entrenamiento y la validación usando los datos de test) sean realmente fiables.

El mejor modelo basado en los datos de entrenamiento, se puede extraer como una lista que incorpora el objeto de clase `caret::train` con el mejor modelo, junto con información adicional, empleando la función `methylearning::get_best_model`. Para obtener lo mismo, pero para los datos de test, se puede emplear la función `methylearning::get_best_model_test`.

```
# Mejor modelo basándose en la validación de los datos de entrenamiento.
get_best_model(ml_c)
```

```
$best_model
Random Forest

36 samples
20 predictors
 2 classes: 'in.remission', 'went.on.to.relapse'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 1 times)
Summary of sample sizes: 32, 32, 32, 33, 32, 33, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   2    0.8333333  0.64
  11    0.7750000  0.53
  20    0.7333333  0.43

Kappa was used to select the optimal model using the largest value.
The final value used for the model was mtry = 2.

$best_fs
[1] "limma_fs"

$best_cls
[1] "rf"

$best_acc
```

```
[1] 0.8333333

$best_kpp
[1] 0.64
```

También se puede obtener la lista con la selección de variables que mejor ha funcionado, tanto en la validación de los datos de entrenamiento `methylearning::get_best_feature_set`, como en la evaluación de los datos de test `methylearning::get_best_feature_set_test`.

```
# Obtener el conjunto de marcadores que han funcionado mejor
# en la validación de los datos de entrenamiento.
get_best_feature_set(ml_c)
```

```
 [1] "cg18003698" "cg15722265" "cg23420331" "cg05822031" "cg25506432"
   "cg11343579"
 [7] "cg11313124" "cg26394916" "cg13403615" "cg11125805" "cg00584602"
   "cg04772797"
[13] "cg00112048" "cg02625024" "cg16105754" "cg05735590" "cg01056135"
   "cg20575848"
[19] "cg04258333" "cg16734526"
```

Si los datos se particionaron y se realizó la evaluación de los datos de test, se puede obtener una matriz de confusión para el resultado obtenido, de la combinación de algoritmo de FS y clasificación deseado. Por ejemplo, para obtener la matriz de confusión de las predicciones de los datos de test realizadas por el modelo Boruta + kNN:

```
get_confusionMatrix_test(ml_c, fs_method = "boruta_fs", cls_method = "knn")
```

```
Confusion Matrix and Statistics

                    Reference
Prediction           in.remission went.on.to.relapse
  in.remission                  7                  0
  went.on.to.relapse            1                  8

               Accuracy : 0.9375
                 95% CI : (0.6977, 0.9984)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 0.0002594
```

```
                 Kappa : 0.875
 Mcnemar's Test P-Value : 1.0000000

            Sensitivity : 0.8750
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.8889
             Prevalence : 0.5000
         Detection Rate : 0.4375
   Detection Prevalence : 0.4375
      Balanced Accuracy : 0.9375

       'Positive' Class : in.remission
```

En cuanto a la exploración gráfica de los resultados, se ofrecen muchas posibilidades. Por ejemplo, la función `plot`, aplicada al objeto de clase `ml_cls`, devuelve un diagrama de barras con los resultados de `Accuracy` del mejor modelo obtenido por validación de los datos de entrenamiento.

```
plot(ml_c)
```

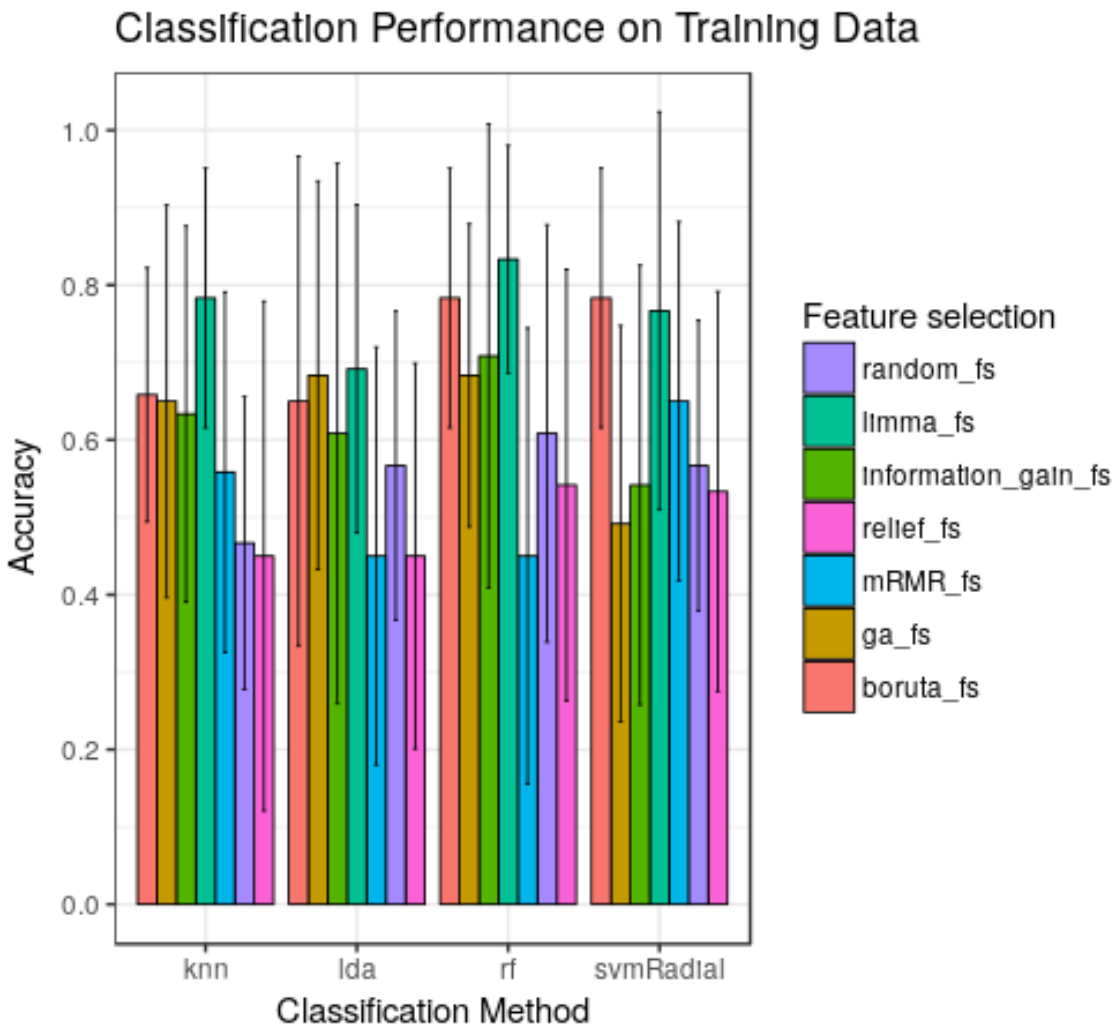

Figura 2.13

Ajustando los argumentos de `plot` se pueden obtener varios gráficos diferentes.

```
# Diagrama de barras para el valor Kappa del mejor modelo por validación
# de los datos de entrenamiento.
plot(ml_c, mode = "Kappa")
```

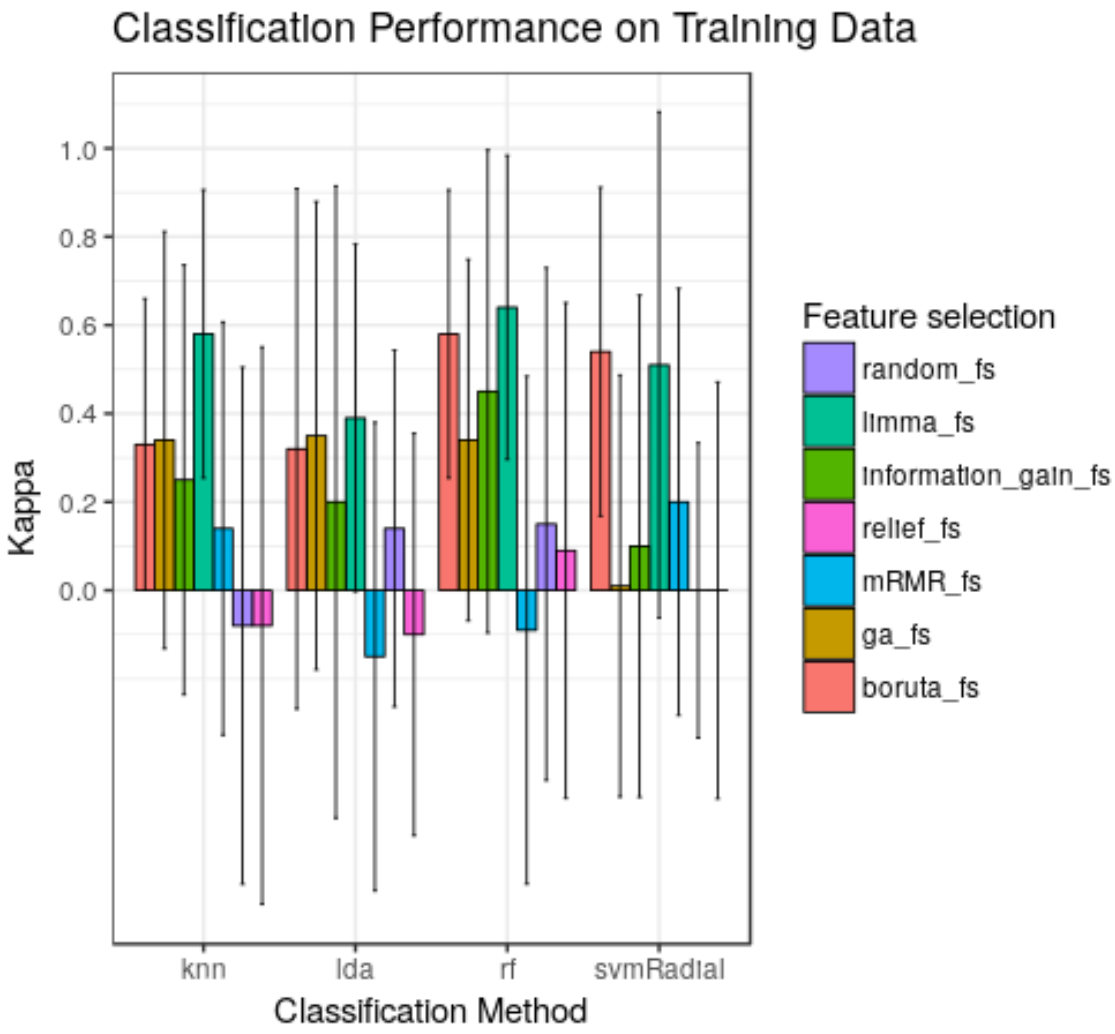

Figura 2.14

Para obtener gráficas similares, con los resultados de *Accuracy* y *Kappa*, pero de las predicciones con los datos de test, se puede emplear la función específica `methylearning::plot_test`, que funciona de manera muy similar a `plot`.

Un tipo de gráfico muy interesante, posible sólo cuando el problema de clasificación implica únicamente dos clases, son las curvas ROC.

```
# ROC curves para los mejores modelos de entrenamiento seleccionados.
plot(ml_c, mode = "ROC", fs_ROC = "boruta_fs", cls_ROC = "knn")
plot(ml_c, mode = "ROC", fs_ROC = "limma_fs", cls_ROC = "rf")
```

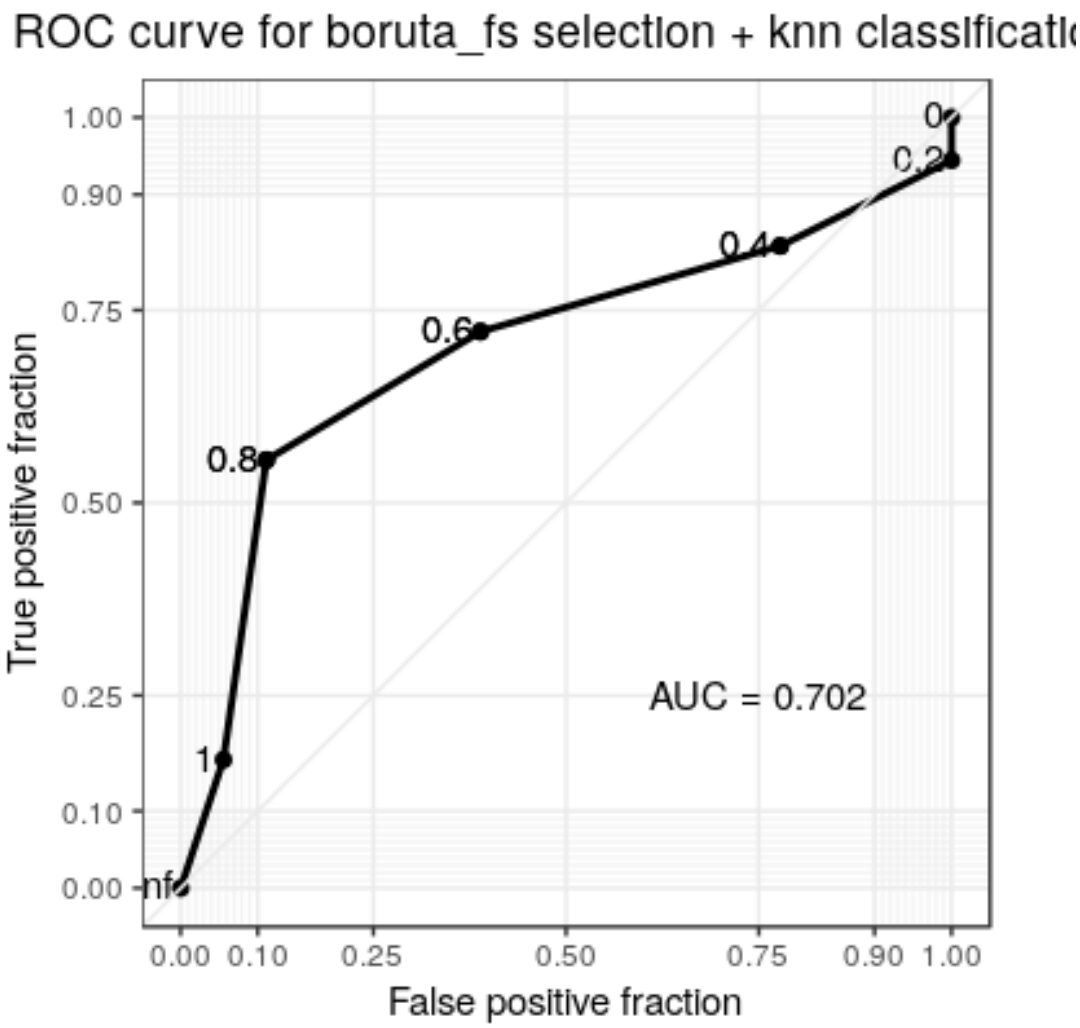
ROC curve for boruta_fs selection + knn classificatio
True positive fraction
False positive fraction
1.00
0.90
0.75
0.50
0.25
0.10
0.00
0.00
0.10
0.25
0.50
0.75
0.90
1.00
0.2
0.4
0.6
0.8
1
AUC = 0.702

Figura 2.15

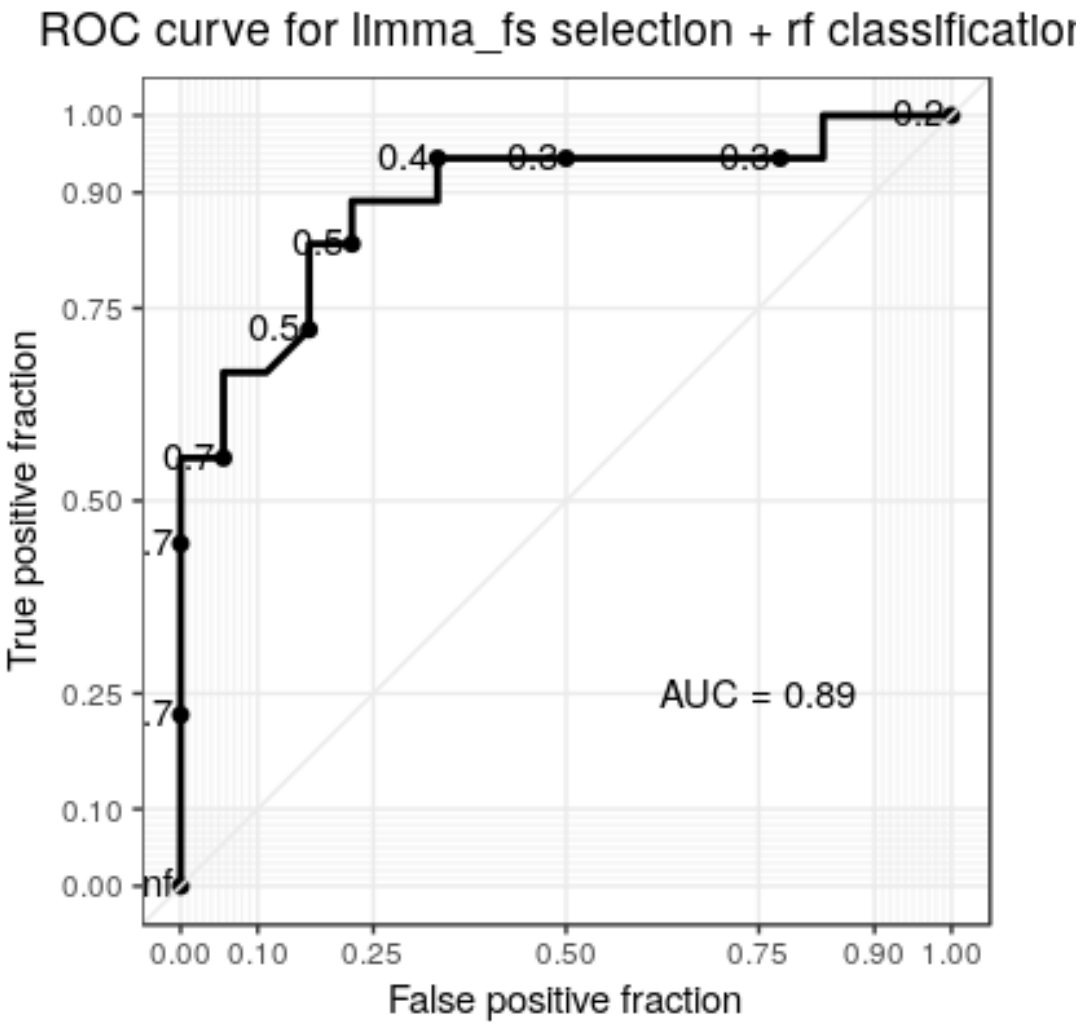

Figura 2.16

Por último, aunque no menos importante. También se facilitan gráficas del tiempo empleado por cada algoritmo de clasificación, por separado o sumado FS + clasificación.

```
# Tiempo de computación de la fase de clasificación.
plot(ml_c, mode = "Time", time_mode = "cls_only")
# Tiempo de computación total por combinación.
plot(ml_c, mode = "Time")
```

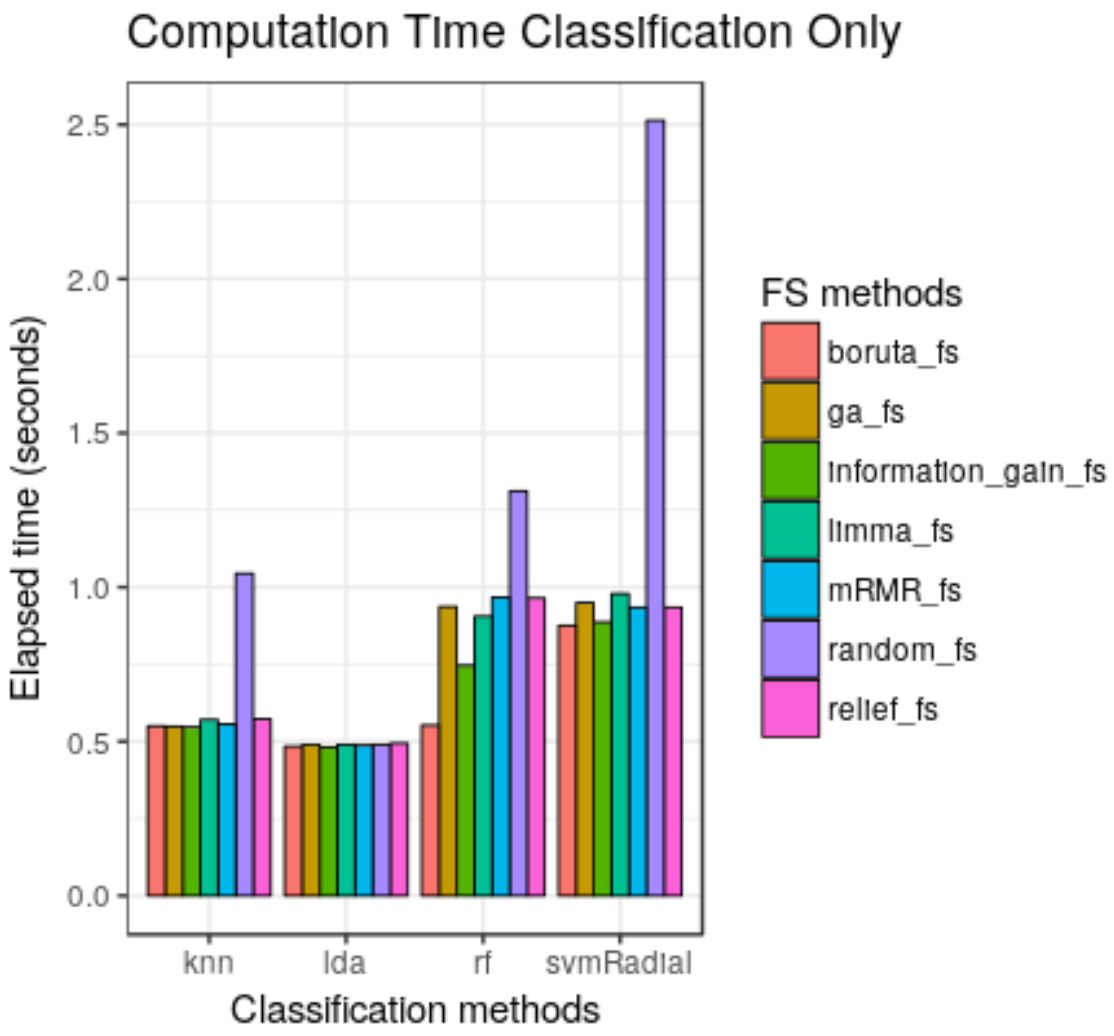
Computation Time Classification Only
Elapsed time (seconds)
2.5
2.0
1.5
1.0
0.5
0.0
knn
lda
rf
svmRadial
Classification methods
FS methods
boruta_fs
ga_fs
Information_gain_fs
limma_fs
mRMR_fs
random_fs
relief_fs

Figura 2.17

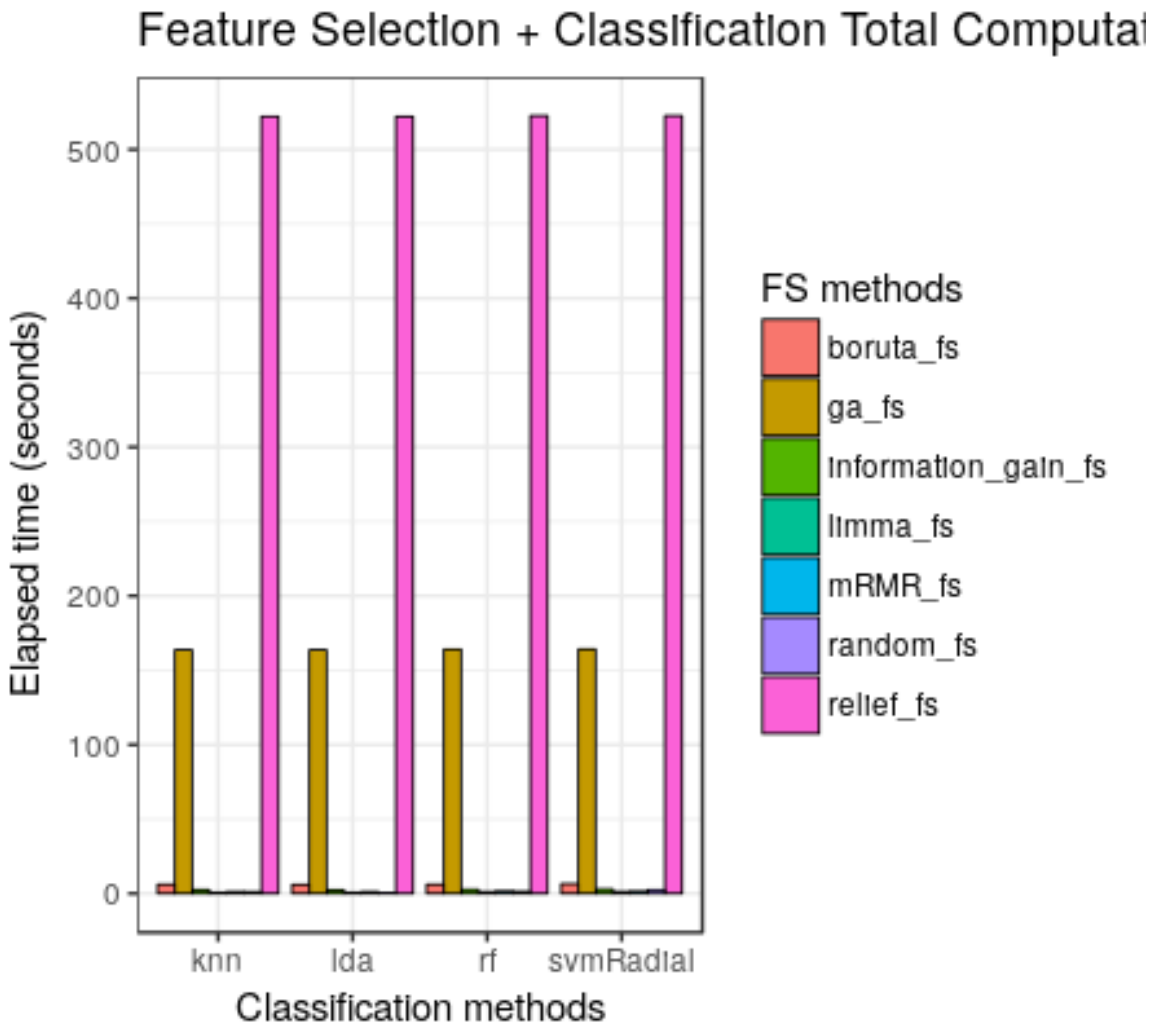

Figura 2.18

Se puede comprobar como el set de datos aleatorios parece dificultar el cálculo de todos los algoritmos de clasificación, excepto “lda”. De todos modos, en el cómputo total del tiempo empleado destaca la larga fase de FS que arrastran los métodos “relief” y “ga”.

2.4. Discusión

En el apartado anterior se ha visto un caso de uso de la herramienta para un set de datos reales. Como se puede comprobar, el paquete es muy sencillo de usar y

los análisis son muy directos y claros. Aporta un valor añadido claro, que se podría resumir en los siguientes puntos:

1. Agrupa muchas funciones existentes en otros paquetes en un único entorno homogéneo, lo que agiliza el proceso de creación de modelos.

2. Los datos quedan contenidos en tres tipos de objetos claramente definidos, con métodos propios que permiten extraer muchos datos.

3. Permite probar, de manera casi automática, múltiples combinaciones de algoritmos de FS y de clasificación y evaluar su rendimiento conjunto, tanto a nivel de rendimiento del modelo propiamente, como del tiempo de computación empleado.

4. Da la posibilidad de ajustar los parámetros más importantes en cada aspecto del análisis, sin caer en un exceso de opciones.

5. Emplea algoritmos de probada eficacia e implementaciones ampliamente conocidas por la comunidad. Eso facilita la comprensión de los resultados obtenidos, a la vez que asegura su calidad.

6. Implementa modificaciones de ciertos algoritmos de FS que exploran metodologías no tratadas por los otros algoritmos, ampliando el espectro de uso del paquete.

Del mismo modo, también es de destacar el valor añadido que aporta la aplicación web asociada al paquete `methylearning`, facilitando y agilizando todas las tareas. En la experiencia de uso real de la herramienta, se comprueba que el disponer de todos los menús con su múltiples opciones descarga al investigador de recordar un gran número de variables y parámetros, lo que redunda en un análisis más fluido. Además, es capaz de aportar prácticamente todas las funcionalidades del paquete `methylearning`, por lo que se puede hacer uso de esta aplicación *Shiny* sin perder calidad ni versatilidad en el análisis.

Para demostrar la utilidad del paquete `methylearning`, los resultados obtenidos en el caso práctico expuesto (ver sección 2.3) son muy ilustrativos. Los autores del artículo original [58] estaban interesados en identificar sitios CpG cuya metilación diferencial pudiera predecir la recaída de los pacientes con ALL. Como resultado de sus análisis, encontraron que los patrones de metilación de los pacientes estaban fuertemente correlacionados con el subtipo genético de cada muestra, fallando en el objetivo de encontrar predictores para la recaída tras el tratamiento. En el análisis de demostración que se incluye en la presente memoria (ver sección 2.3) se puede comprobar como, a pesar de haber reducido el set de datos inicial a sólo 1000 CpGs

y particionar los datos previamente, hemos obtenido un modelo que tiene cierta capacidad de discriminación, como se ve en sus estadísticas:

	Datos de entrenamiento	Datos de test
Accuracy	0.83	0.75
Kappa	0.64	0.50

Cuadro 2.2: Rendimiento del modelo limma + rf

Teniendo en cuenta el carácter demostrativo del análisi, el rendimiento del modelo no es despreciable[2]. Así, si bien los autores del trabajo original pueden estar en lo correcto al afirmar que la mayor fuente de variabilidad en sus muestras es debida al subtipo genético de cada muestra y que ello dificulta el encontrar biomarcadores claros, sus datos sí parecen ofrecer la posibilidad de generar un modelo de clasificación con utilidad real. Por supuesto, la construcción de un modelo de clasificación válido en el entorno clínico requeriría de la validación de los resultados obtenidos en, al menos, una cohorte independiente de pacientes. A pesar de haber validado el modelo por CV, el sobreajuste siempre se da en mayor o menor medida, además de que es posible que la cohorte empleada en el artículo no sea representativa de la población general, un extremo que una cohorte independiente ayudaría a identificar.

En lo que respecta a la capacidad de analizar el diferente rendimiento de cada algoritmo, el software desarrollado ofrece ventajas evidentes y, a pesar de que harían falta muchas más pruebas con sets de datos diferentes, ya se pueden extraer algunas conclusiones. Centrando la atención en los algoritmos de FS, sorprende comprobar que los dos más lentos son del tipo *filter* y, en el caso concreto del algoritmo "correlation based", muchísimo más lento que "information gain", que selecciona las mismas variables. Por las pruebas internas durante el desarrollo del proyecto, estos dos algoritmos tienden a dar unas listas de variables muy parecidas, siendo "information gain" mucho más rápido. Mención especial merece uno de los algoritmos diseñados específicamente para este proyecto, "mseq" (selección secuencial de variables, modificado), que resulta excesivamente lento para sets de datos de más de 500 variables. También se ha observado que los métodos "anova" y "lima" resultan muy similares en cuanto a sus selecciones, algo lógico ya que su mecanismo es muy similar. En cuanto al rendimiento de los clasificadores, se ajustan bastante a lo que se observa en la literatura. Sorprende comprobar lo estrechamente ligado que está su rendimiento a la selección de variables anterior. Como ejemplo, en el caso práctico expuesto se puede ver que el mejor modelo con los datos de entrenamiento,

[2]Pruebas internas empleando más variables han llegado fácilmente a una *Accuracy* y *Kappa* de 1

“rf”, lo es sólo con la selección de variables realizada por “limma”. Así, con la FS por el método “mRMR”, resulta de los peores. En este sentido, los algoritmos de FS que aportan resultados más consistentes con todos los clasificadores serían “limma” y “Boruta”, este último, habiendo seleccionado solamente 2 CpGs. Esto convierte a “Boruta” en uno de los mejores algoritmos (de los probados) para encontrar biomarcadores. Finalmente, al igual que para los algoritmos de FS, los algoritmos de clasificación también tienen grandes diferencias en cuanto a velocidad de ejecución, aunque no llegan a ser tan extremas como en el caso de los algoritmos de FS.

Capítulo 3

Conclusiones

En la realización del presente proyecto se ha profundizado en todos los aspectos de creación de un modelo de clasificación: la elección, exploración y preparación de los datos de entrada, la selección de variables para reducir la dimensionalidad, la elección del modelo de clasificación adecuado. Sin olvidar las interconexiones entre cada una de las etapas, donde los resultados obtenidos en una etapa tienen efectos evidentes en las siguientes. Además, también se ha adquirido mucha soltura en técnicas de programación en lenguaje `R`, diseño de algoritmos, elaboración de paquetes para la distribución del software, documentación del mismo y elaboración de aplicaciones web mediante la herramienta *Shiny*. Finalmente, aspectos técnicos a parte, la elaboración de un proyecto tan amplio en un tiempo tan limitado también ha supuesto un reto que ha hecho necesario una concienzuda planificación del trabajo.

En lo que respecta a los logros propuestos y los resultados obtenidos, el presente trabajo logra sus objetivos de ofrecer una herramienta que facilite el uso de técnicas de ML, tanto de FS como de clasificación, a la vez que proporciona múltiples medios para la evaluación de los resultados obtenidos. Adicionalmente, también se proporciona una interfaz web de uso más sencillo, que aún así da acceso a todo el potencial del paquete `methylearning`.

Sin embargo, los recursos computacionales necesarios para realizar análisis en un entorno real de trabajo son realmente altos. Como muestra, la dimensionalidad de los datos ha tenido que ser reducida desde un principio para poder comenzar a trabajar, lo que sin duda ha condicionado todos los resultados posteriores. Si bien las herramientas creadas ofrecen un *farmework* completo, sencillo y homogéneo, el coste computacional de cada una de sus partes se suma. Ofrecer tantos algoritmos y, algunos de ellos, tan costosos hace que ejecutar el análisis por defecto, el más exhaustivo, sea una tarea solo al alcance de grandes estaciones de trabajo. El alto coste computacional ya era una problemática que se preveía desde el principio y

los intentos de mitigación, como por ejemplo la paralelización de algunas tareas clave, han aportado también sus propios problemas, en forma de mayor consumo de memoria. Por lo tanto, los requerimientos de hardware necesarios para ejecutar, tanto el paquete `methylearning` como la aplicación *Shiny* asociada, limitan el uso de la herramienta, si bien, cabe destacar que herramientas de enfoque tan amplio y exhaustivo como la presentada aquí acostumbran a adolecer de los mismos problemas. Como punto añadido, la duración de algunos cálculos también ha condicionado el desarrollo de la aplicación *Shiny*, eliminando la mayoría de la reactividad automática de la que hacen gala normalmente estas aplicaciones.

Uno de los mayores problemas que han dificultado el proyecto desde el principio ha sido la falta de tiempo. En un principio, se infraestimó la cantidad de tiempo necesaria para completar un proyecto de las características del aquí presentado. Afortunadamente, tras el primer informe de seguimiento, se revisó todo el calendario y se aumento el número de horas de dedicación, lo que consiguió poner el proyecto al día y aseguró su correcta conclusión.

Precisamente por cuestiones de tiempo, algunos aspectos se han dejado como mejoras futuras. Por un lado, la interfaz gráfica propuesta en la aplicación *Shiny*, si bien es funcional, resulta demasiado escasa en detalles y poco intuitiva para un primer contacto, si no se conocen todas las funcionalidades del paquete `methylearning` de antemano. En los que respecta al paquete en sí, existe también mucho margen de mejora. Por ejemplo, en relación al rendimiento en tiempo de ejecución, la dependencia de un gran número de paquetes de `R`, muchos programados en este mismo lenguaje, lastra el rendimiento. También es de destacar que la dependencia del paquete `caret`, algo necesario por cuestiones de tiempo y efectividad, resta algo de flexibilidad al conjunto de la herramienta. Finalmente, la velocidad de ejecución del algoritmo “mseq”, diseñado expresamente para este proyecto, lo convierten en poco práctico y hacen necesario una recodificación del mismo, quizá programándolo directamente en un lenguaje más eficiente, como `C` y replanteando el diseño del algoritmo para agilizar su ejecución.

Capítulo 4

Glosario

5mC: 5-metilcitosina.

ALL: del inglés *acute lymphoblastic leukemia*, leucemia linfoblástica aguda.

ANN: *Artificial Neural Networks*.

ANOVA: análisis de la varianza.

BS-seq: del inglés *Bisulfite-sequencing*, se refiere a la secuenciación de bisulfito, también conocida como WGBS.

CFS: *Correlation-based Feature Selection*, selección de marcadores en base a su correlación.

CpG: dinucleótido Citosina y Guanina. Su unión en una monohebra de DNA se indica mediante la "p", que denota el enlace fosfato entre ambos nucleótidos.

CPUs: del inglés Central Processor Units, unidades de procesamiento central.

CV: del inglés *cross-validation*, validación cruzada.

DNA: del inglés *deoxyribonucleic acid*, ácido desoxiribonucleico.

ENCODE: *Encyclopedia of DNA Elements*.

FS: del inglés *Feature Selection*, selección de variables o marcadores.

GEO: *Gene Expression Omnibus*.

k-NN: *k-Nearest Neighbor*.

lasso: *least absolute shrinkage and selection operator*.

LDA: *Linear Discriminant Analysis*.

LOOCV: del inglés, *leave-one-out cross-validation*.

MeDIP: del inglés, *methylated DNA immunoprecipitation*, es una técnica de inmunoprecipitación de 5mC usando anticuerpos específicos.

ML: del inglés *machine learning*, aprendizaje automático mediante algoritmos y métodos computacionales, que permiten realizar tareas de clasificación y/o predicción (regresión).

MLP: *Multi Layer Perceptron*.

MMH: del inglés *Maximum Margin Hyperplane*, Hiperplano de Margen Máximo .

mRNA: RNA mensajero.

mRMR: *minimum Redundancy Maximum Relevance*.

NCBI: *National Center for Biotechnology Information*.

NGS: del inglés *Next Generation Sequencing*, conjunto de técnicas de secuenciación masivas, con un rendimiento en cuanto a número de secuencias obtenidas muy superior a la técnica de secuenciación de Sanger.

PCA: análisis de componentes principales.

RAM: del inglés *Random Access Memory*, memoria de acceso aleatorio.

RBF: *Radial Basis Function*.

RNA: del inglés *ribonucleic acid*, ácido ribonucleico.

ROC: *receiver operating characteristic curv*.

RRBS: del inglés *Reduced Representation Bisulfite Sequencing*, es una técnica de secuenciación de bisulfito que emplea encimas de restricción.

SNV: del inglés *Single Nucleotide Variation*, variación de un único nucleótido, engloba tanto mutaciones como polimorfismos poblacionales.

SVM: del inglés Support Vector Machine, máquina de soporte vectorial.

SVM-RFE: del inglés *recursive feature elimination* with *support vector machines*, eliminación recursiva de variables empleando máquinas de soporte vectorial.

TCGA: *The Cancer Genome Atlas*.

UTR: del inglés *untranslated region*, región del mRNA no traducida a proteína.

WGBS: del inglés *Whole Genome Bisulfite Sequencing*, secuenciación del genoma completo previamente tratado con bisulfito sódico.

Capítulo 5

Bibliografía

[1] Fei Jiang, Yong Jiang, Hui Zhi, Yi Dong, Hao Li, Sufeng Ma, Yilong Wang, Qiang Dong, Haipeng Shen, and Yongjun Wang. Artificial intelligence in healthcare: past, present and future. *Stroke and Vascular Neurology*, 2017.

[2] Alexander L. Fogel and Joseph C. Kvedar. Artificial intelligence powers digital medicine. *npj Digital Medicine*, 1, 2018.

[3] Xinyu Zhang, Ying Hu, Amy Justice, Boyang Li, Zuoheng Wang, Hongyu Zhao, John Krystal, and Ke Xu. Dna methylation signatures analysis with illumina infinitum methylationepic and infinium human methylation 450k beadchip. *Protocol Exchange*, Nov 2017.

[4] Stephen J. Clark, Sébastien A. Smallwood, Heather J. Lee, Felix Krueger, Wolf Reik, and Gavin Kelsey. Genome-wide base-resolution mapping of dna methylation in single cells using single-cell bisulfite sequencing (scbs-seq). *Nature Protocols*, 12:534 EP –, Feb 2017.

[5] Lawrence B. Holder, M. Muksitul Haque, and Michael K. Skinner. Machine learning for epigenetics and future medical applications. *Epigenetics*, 12(7):505–514, 2017. PMID: 28524769.

[6] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018.

[7] Winston Chang, Joe Cheng, JJ Allaire, Yihui Xie, and Jonathan McPherson. *shiny: Web Application Framework for R*, 2017. R package version 1.0.5.

[8] Matthew E Ritchie, Belinda Phipson, Di Wu, Yifang Hu, Charity W Law, Wei Shi, and Gordon K Smyth. limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research*, 43(7):e47, 2015.

[9] Max Kuhn. caret: Classification and regression training, 2018. R package version 6.0-80.

[10] Dmitry V. Fyodorov, Bing-Rui Zhou, Arthur I. Skoultchi, and Yawen Bai. Emerging roles of linker histones in regulating chromatin structure and function. *Nature Reviews Molecular Cell Biology*, 19(192), 10 2017.

[11] A. Wolffe. Chapter two - chromatin structure. In A. Wolffe, editor, *Chromatin (Third Edition)*, pages 7 – 172. Academic Press, London, third edition edition, 2000.

[12] Howard Cedar and Yehudit Bergman. Programming of dna methylation patterns. *Annual Review of Biochemistry*, 81(1):97–117, 2012. PMID: 22404632.

[13] Zachary D. Smith and Alexander Meissner. Dna methylation: roles in mammalian development. *Nature Reviews Genetics*, 14(3):204–20, 03 2013.

[14] Andrews Andrew J. and Luger Karolin. *Histone Modifications: Chemistry and Structural Consequences*, pages 1–10. American Cancer Society, 2008.

[15] Vichithra R B Liyanage, Jessica S Jarmasz, Nanditha Murugeshan, Marc R Del Bigio, Mojgan Rastegar, and James R Davie. Dna modifications: Function and applications in normal and disease states. *Biology (Basel)*, 3(4):670–723, 12 2014.

[16] A Razin and H Cedar. Dna methylation and gene expression. *Microbiological Reviews*, 55(3):451–458, 09 1991.

[17] Da-Hai Yu, Carol Ware, and Robert A et al. Waterland. Developmentally programmed 3' cpg island methylation confers tissue- and cell-type-specific transcriptional activation. *Molecular and Cellular Biology*, 33(9):1845–1858, 05 2013.

[18] Azam Moosavi and Ali Motevalizadeh Ardekani. Role of epigenetics in biology and human diseases. *Iranian Biomedical Journal*, 20(5):246–258, 11 2016.

[19] Holger Heyn, Jesús Méndez-González, and Manel Esteller. Epigenetic profiling joins personalized cancer medicine. *Expert Review of Molecular Diagnostics*, 13(5):473–479, 2013.

[20] Ruth Pidsley, Elena Zotenko, and Timothy J et al. Peters. Critical evaluation of the illumina methylationepic beadchip microarray for whole-genome dna methylation profiling. *Genome Biology*, 17:208, 10 2016.

[21] Pan Du, Xiao Zhang, Chiang-Ching Huang, Nadereh Jafari, Warren A. Kibbe, Lifang Hou, and Simon M. Lin. Comparison of beta-value and m-value methods for quantifying methylation levels by microarray analysis. *BMC Bioinformatics*, 11(1):587, Nov 2010.

[22] Daniel J. Weisenberger, David Van Den Berg adn Fei Pan, Benjamin P. Berman, and Peter W. Laird. Comprehensive dna methylation analysis on the illumina infinium assay platform. Technical report, Illumina, 3 2008.

[23] Illumina. Infinium humanmethylation450 beadchip. Technical report, Illumina, 3 2012.

[24] Illumina. Infinium methylationepic beadchip. Technical report, Illumina, 2015.

[25] Ryan Lister, Mattia Pelizzola, and Robert H. et al. Dowen. Human dna methylomes at base resolution show widespread epigenomic differences. *Nature*, 462(7271):315–322, 11 2009.

[26] Hongcang Gu, Zachary D Smith, Christoph Bock, Patrick Boyle, Andreas Gnirke, and Alexander Meissner. Preparation of reduced representation bisulfite sequencing libraries for genome-scale dna methylation profiling. *Nature Protocols*, 6(4):468–81, 03 2011.

[27] Thomas A Down, Vardhman K Rakyan, and Daniel J et al. Turner. A bayesian deconvolution strategy for immunoprecipitation-based dna methylome analysis. *Nature Biotechnology*, 26(7):779–85, 07 2008.

[28] Christof Angermueller, Heather J. Lee, Wolf Reik, and Oliver Stegle. Deepcpg: accurate prediction of single-cell dna methylation states using deep learning. *Genome Biology*, 18(1):67, Apr 2017.

[29] Qian Peng and Joseph R. Ecker. Detection of allele-specific methylation through a generalized heterogeneous epigenome model. *Bioinformatics*, 28(12):i163–i171, 2012.

[30] Athina Vidaki, David Ballard, Anastasia Aliferi, Thomas H. Miller, Leon P. Barron, and Denise Syndercombe Court. Dna methylation-based forensic age prediction using artificial neural networks and next generation sequencing. *Forensic Science International: Genetics*, 28:225–236, 05 2017.

[31] M. Muksitul Haque, Lawrence B. Holder, and Michael K. Skinner. Genome-wide locations of potential epimutations associated with environmentally induced epigenetic transgenerational inheritance of disease using a sequential machine learning prediction approach. *PLOS ONE*, 10(11):1–25, 11 2015.

[32] Péter Adorján, Jürgen Distler, and Evelyne et al. Lipscher. Tumour class prediction and discovery by microarray-based dna methylation analysis. *Nucleic Acids Research*, 30(5):e21, 03 2002.

[33] Zhihua Cai, Dong Xu, Qing Zhang, Jiexia Zhang, Sai-Ming Ngai, and Jianlin Shao. Classification of lung cancer using ensemble-based feature selection and machine learning methods. *Mol. BioSyst.*, 11:791–800, 2015.

[34] Sebastian Moran, Anna Martínez-Cardús, and Sergi et al. Sayols. Epigenetic profiling to classify cancer of unknown primary: a multicentre, retrospective analysis. *The Lancet Oncology*, 17(10):1386 – 1395, 10 2016.

[35] Katherine A. Hoadley, Christina Yau, and Toshinori et al. Hinoue. Cell-of-origin patterns dominate the molecular classification of 10,000 tumors from 33 types of cancer. *Cell*, 173(2):291 – 304.e6, 04 2018.

[36] Brett Lantz. *Machine Learning with R*. Packt Publishing, 2nd edition, 2015.

[37] Robert Lowe and Vardhman K. Rakyan. Correcting for cell-type composition bias in epigenome-wide association studies. *Genome Medicine*, 6(3):23, Mar 2014.

[38] V. Bolón-Canedo, N. Sánchez-Maroño, A. Alonso-Betanzos, J.M. Benítez, and F. Herrera. A review of microarray datasets and applied feature selection methods. *Information Sciences*, 282:111 – 135, 2014.

[39] Zena Hira and Duncan F. Gillies. A review of feature selection and feature extraction methods applied on microarray data. *Advances in Bioinformatics*, 2015:1–13, 07 2015.

[40] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[41] P. R. Anukrishna and V. Paul. A review on feature selection for high dimensional data. In *2017 International Conference on Inventive Systems and Control (ICISC)*, pages 1–4, Jan 2017.

[42] Mark A. Hall. *Correlation-based Feature Selection for Machine Learning*. PhD thesis, The University of Waikato, Hamilton, NewZeland, 4 1999. An optional note.

[43] Kenji Kira and Larry A. Rendell. The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pages 129–134. AAAI Press, 1992.

[44] Igor Kononenko. Estimating attributes: Analysis and extensions of relief. In Francesco Bergadano and Luc De Raedt, editors, *Machine Learning: ECML-94*, pages 171–182, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[45] Hanchuan Peng, Fuhui Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, Aug 2005.

[46] Iñaki Inza, Basilio Sierra, and Rosa Blanco. Gene selection by sequential search wrapper approaches in microarray cancer class prediction. *Journal of Intelligent and Fuzzy Systems*, 12:25–33, 01 2002.

[47] Maria Fernanda Wanderley, Vincent Gardeux, René Natowicz, and Antônio P. Braga. Ga-kde-bayes: An evolutionary wrapper method based on non-parametric density estimation applied to bioinformatics problems. In *21st European Symposium on Artificial Neural Networks-ESANN*, 04 2013.

[48] Miron Kursa and Witold Rudnicki. Feature selection with the boruta package. *Journal of Statistical Software, Articles*, 36(11):1–13, 2010.

[49] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1):389–422, Jan 2002.

[50] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[51] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[52] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[53] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[54] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

[55] FISHER R. A. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[56] C. Radhakrishna Rao. The utilization of multiple measurements in problems of biological classification. *Journal of the Royal Statistical Society. Series B (Methodological)*, 10(2):159–203, 1948.

[57] Sean Davis and Paul Meltzer. Geoquery: a bridge between the gene expression omnibus (geo) and bioconductor. *Bioinformatics*, 14:1846–1847, 2007.

[58] Alem S Gabriel, Fadhel M Lafta, Edward C Schwalbe, Sirintra Nakjang, Simon J Cockell, Alice Iliasova, Amir Enshaei, Claire Schwab, Vikki Rand, Steven C Clifford, Sally E Kinsey, Chris D Mitchell, Ajay Vora, Christine J Harrison, Anthony V Moorman, and Gordon Strathdee. Epigenetic landscape correlates with genetic subtype but does not predict outcome in childhood acute lymphoblastic leukemia. *Epigenetics*, 10(8):717–726, 2015. PMID: 26237075.