Machine learning para la selección de genes implicados en el desarrollo de *Arabidopsis thaliana* utilizando datos de expresión génica

Maria Teresa Saura Sánchez
Máster universitario en Bioinformática y Bioestadística (UOC-UB)
Bioinformática y bioestadística

Consultor - Esteban Vegas Lozano
Profesor responsable de la asignatura - Alexandre Sánchez Pla

Enero 2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Machine learning para la selección de genes implicados en el desarrollo de Arabidopsis thaliana utilizando datos de expresión génica*
Nombre del autor:	*Maria Teresa Saura Sánchez*
Nombre del consultor/a:	*Esteban Vegas Lozano*
Nombre del PRA:	*Alexandre Sánchez Plá*
Fecha de entrega (mm/aaaa):	01/2019
Titulación::	*Máster universitario en Bioinformática y Bioestadística (UOC-UB)*
Área del Trabajo Final:	*Bioinformática y bioestadística*
Idioma del trabajo:	*Castellano*
Palabras clave	*Machine learning, DNA microarray, plant development*

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

Las plantas desarrollan nuevos órganos y tejidos a lo largo de su ciclo de vida. Gracias a la tecnología de microarrays de ADN se ha podido indagar en los mecanismos transcripcionales asociados al desarrollo de estas estructuras en la especie modelo *Arabidopsis thaliana*. Sin embargo, los estudios publicados analizan un reducido número de muestras para la selección de genes de desarrollo según la expresión específica de éstos en los distintos tejidos de la planta, sin tener en cuenta relaciones más complejas entre los genes. En este trabajo se propone el uso de técnicas de *machine learning* para la selección de genes relevantes en el desarrollo de las estructuras de la planta *A. thaliana*. Para ello, se generó una base de datos con más de 500 perfiles de expresión correspondientes a semillas, plántulas, raíces, hojas y flores. Sobre estos datos se utilizaron tres métodos de *machine learning* para la selección de genes según su importancia en modelos de clasificación: FP-RF, RF-RFE SVM-RFE. Además, se evaluó la capacidad del método autoencoder para la representación de estos datos en una dimensión reducida. Los grupos de genes seleccionados por ML mostraron un alto rendimiento en la clasificación de las muestras por RF, SVM y ANN. Asimismo, el análisis funcional *in silico* de estos genes reveló su implicancia en los procesos de desarrollo de la planta. Estos resultados ponen de manifiesto la potencia de los algoritmos de ML para el estudio del desarrollo de las plantas a partir de datos de expresión génica.

Abstract (in English, 250 words or less):

Transcriptional programs are important in the development of the structures throughout the life cycle of plants. DNA microarray technology has provided a useful tool to discover relevant genes in the development of the reference plant *Arabidopsis thaliana*. However, previous studies use a reduced number samples to discover marker genes based on its specific expression along the tissues. In this work, a machine learning approach is presented to select relevant genes in the development of *A. Thaliana*. A database was built with more than 500 expression profiles corresponding to seeds, seedlings, roots, leaves and flowers. Gene selection was carried out with three different ML methods: FP-RF, RF-RFE, SVM-RFE. Furthermore, an autoencoder architecture was evaluated for dimensionality reduction of the data. The genes selected by ML techniques yield high classification performance in SVM, RF and ANN algorithms. Moreover, these genes are biologically relevant to plant development proccess. This work provides a new approach to study plant development from gene expression data.

Índice

Lista de figuras

Lista de tablas

Capítulo 1

Introducción

1.1 Contexto y justificación del Trabajo

1.1.1 Programas transcripcionales en el desarrollo de *Arabidopsis thaliana.*

1.1.1.1 Desarrollo de las plantas

Las plantas son organismos sésiles que a diferencia de la mayoría de los animales desarrollan nuevos tejidos a lo largo de su ciclo de vida. Estos tejidos conforman estructuras especializadas que les permiten a las plantas la adaptación a las condiciones ambientales en las que crecen además de cumplir las funciones características de la edad de la planta.

Una de las especies más usadas como organismo modelo para el estudio de las plantas es *Arabidopsis thaliana* [1]. *A. thaliana* es una planta dicotiledónea que presenta dos fases diferenciadas a lo largo de su desarrollo [2]. La primera es una fase vegetativa en la que la planta desarrolla hojas y raíces que le permiten crecer adquiriendo biomasa. La segunda fase comprende la fase reproductiva que se induce con las condiciones de fotoperiodo y temperatura adecuadas y que permiten a la planta desarrollar flores que darán lugar a su progenie.

El ciclo de vida de *A. thaliana* comienza a partir de la germinación de la semilla que dará lugar a un nuevo individuo. Tras la germinación, si las condiciones ambientales son óptimas, se da el establecimiento de la plántula en la que se desarrolla la radícula y los cotiledones. Estos últimos contienen los metabolitos necesarios para seguir el desarrollo de la planta. Tras el establecimiento de la plántula se desarrollan las primeras hojas de la planta en la parte aérea y la raíz primaria comienza a desarrollar las raíces secundarias, comenzando la fase vegetativa de la planta. Durante esta fase van emergiendo nuevas hojas y las raíces van creciendo adquiriendo mayor biomasa. Cuando se dan las condiciones necesarias se induce la floración y emerge el tallo principal a partir del cual se desarrollan los tallos secundarios que contienen meristemos florales que darán lugar a las inflorescencias. En cada inflorescencia

se desarrollan múltiples flores compuestas por pétalos, sépalos, anteras y pistilo. El pistilo contiene los óvulos que son fertilizados por el polen presente en las anteras. Tras la fertilización comienza el desarrollo de las semillas inmaduras que son albergadas por el fruto. Finalmente, las semillas maduras se esparcen tras la senescencia del fruto completando el ciclo de vida de la planta.

Las estructuras que componen la planta a lo largo de su vida presentan programas transcripcionales característicos que se mantienen activos durante todo el ciclo por la continua formación de nuevos órganos [3].

1.1.1.2 Microarrays de cDNA para el estudio de la expresión génica en plantas

El estudio de la expresión génica es una herramienta de gran utilidad para comprender los mecanismos moleculares por los que las células vivas ejecutan su función. Para la elucidación de los patrones de expresión en plantas se utilizan principalmente dos técnicas: RNA-seq y *microarrays* de DNA [4,5]. Ambas técnicas consisten en cuantificar los niveles de expresión génica a nivel global con el fin de obtener una visión completa del transcriptoma de un tejido o célula en un momento concreto. La técnica de RNA-seq consiste en el aislamiento de los transcritos y su posterior conversión a DNA complementario (cDNA) para después secuenciar estos fragmentos utilizando plataformas de secuenciación de siguiente generación. Las secuencias obtenidas se mapean en el genoma de referencia y se evalúan los niveles de transcrito correspondientes a cada gen [6]. Aunque la técnica de RNA-seq tiene una mayor potencia y cobertura que la obtenida con los *microarrays* de DNA, su uso en plantas se extendió hace relativamente poco. En el repositorio ArrayExpress (EMBL-EBI) hay disponibles 546 experimentos de RNA-seq en *A. thaliana*, mientras que hay 2453 experimentos obtenidos de *microarrays* para esta especie. Por esto, los datos obtenidos de *microarrays* comprenden una fuente muy valiosa de información para estudios a gran escala.

Por su parte, los *microarrays* se basan en la hibridación del cDNA aislado sobre una plataforma que contiene hebras complementarias representativas del genoma de la especie. Aunque existen distintos tipos de *microarrays* uno de los más populares es el de un único canal. El chip de *microarray* tiene sintetizados en su superfície oligonúcleotidos de ADN complementarios a las secuencias conocidas de los RNA mensajeros. Las muestras de cDNA se hibridan sobre el *microarray* y se emite una señal si se produce la hibridación de un cDNA con su oligonucleótido complementario en el *microarray*.

Una de las plataformas más populares de *microarrays* es Affymetrix [7]. Existen varios chips de Affymetrix diseñados para el estudio del transcriptoma de *A. thaliana*. En concreto, el chip ATH1 contiene más de 22500 sondas correspondientes a unos 24000 genes y es el que tiene mayor cobertura del genoma para esta especie.

1.1.1.3 Patrones de expresión génica en el desarrollo de plantas

Uno de los objetivos principales en el estudio de la biología del desarrollo es elucidar los patrones de expresión asociados a los distintos órganos o estructuras de la planta. En [3]

abarcaron este problema analizando el perfil de expresión de los tejidos de *A. thaliana* en sus distintos estadios de desarrollo utilizando microarrays de DNA. Esto permitió establecer un atlas de expresión que cubre gran parte de los órganos y las estructuras de la planta modelo. Utilizando la misma aproximación se ha estudiado también el transcriptoma asociado a las distintas fases de desarrollo de otras especies como es el caso del arroz o la uva [8,9].

Otros autores han usado diferentes aproximaciones para definir genes específicos asociados a un órgano o estructura concreta. En [10] seleccionan genes específicos del desarrollo de semillas *A. thaliana* comparando la expresión de los genes de semillas con distinto grado de madurez con la expresión de los genes en otras estructuras de las plantas. Con el mismo fin, en [11] utilizan estos datos de expresión y definen genes marcadores del desarrollo de semillas con técnicas de *machine learning*.

1.1.2 *Machine learning* para el estudio de datos de expresión génica

Las técnicas de *machine learning* comprenden un gran número de algoritmos de aprendizaje utilizados ampliamente para resolver problemas diversos y complejos. Debido a su potencial en el análisis de datos de gran dimensión su aplicación también se ha extendido a los datos de expresión génica [12]. En concreto, la clasificación de microarrays y la selección de genes importantes en esta clasificación es una de las áreas más exploradas. Está bien documentada la clasificación de muestras de microarrays para el diagnóstico de distintos tipos de cáncer [13]. También la selección de genes biomarcadores de enfermedades se ha resuelto con éxito utilizando este tipo de algoritmos [14].

1.1.2.1 Algoritmos de clasificación para muestras de microarrays con *machine learning*

Los algoritmos de *machine learning* en problemas de clasificación se dividen principalmente en dos tipos, aprendizaje supervisado y no supervisado. En los esquemas supervisados destacan técnicas como SVM, random forest y redes neuronales en los que las muestras de microarrays se encuentran etiquetadas según la clase a la que pertenecen durante el proceso de aprendizaje [15]. Por su parte, en los métodos no supervisados el aprendizaje se realiza sin previo conocimiento de la pertenencia de las muestras. Tal es el caso de los algoritmos basados en *clusters* como el algoritmo *k-means* [16].

Dentro de estos métodos, los más utilizados en la literatura científica para clasificar muestras de *microarrays* son los del tipo de aprendizaje supervisado. A continuación se detallan las características de algunos de ellos.

Support vector machines

Support vector machines (SVM) consiste en un grupo de técnicas de aprendizaje supervisado usado en problemas de clasificación y regresión [17]. El modelo más simple es el SVM lineal que intenta trazar un hiperplano que separe de forma óptima las muestras que pertenecen a distintas clases. De todos los planos que pueden separar linealmente las muestras en la

distintas clases el algoritmo está diseñado para seleccionar aquel en el margen del hiperplano sea máximo separando las muestras [18]. Se denominan *support vector* a aquellas muestras de cada clase que se encuentran más cercanas a los márgenes del hiperplano.

Este tipo de algoritmo se ha usado ampliamente en la clasificación de patrones de expresión génica. En [19] identifican muestras pertenecientes a pacientes con cáncer a partir de datos de *microarrays*. En [20] utilizan este método para clasificar muestras de *microarrays* de pacientes con Parkinson.

Random forest

Random forest (RF) es un método que utiliza una gran cantidad de árboles de decisión independientes probados sobre conjuntos de datos aleatorios [21]. En el proceso de clasificación, los datos son particionados aleatoriamente y cada grupo genera un árbol de decisión en el que se evalua la capacidad de clasificación de cada uno de estos. La predicción final tiene en cuenta la proporción de árboles que toman una misma decisión. En [22] describen el uso de RF para la clasificación de muestras de *microarrays* de distintos grupos de datos.

Redes neuronales artificiales

Las redes neuronales artificiales (ANN, del inglés *Artificial neural networks*) comprenden un modelo basado en un gran conjunto de neuronas conectadas entre sí, en una analogía al comportamiento del sistema neuronal. Las neuronas están conectadas por enlaces con un peso asociado que multiplica el valor de la neurona de salida. Este método también ha sido explorado para la clasificación de muestras de expresión génica [23,24].

1.1.2.2 Selección de genes con *machine learning.*

La clasificación de datos de expresión génica puede ser un proceso tedioso que requiere una gran potencia computacional por las características de estos datos donde suelen haber una poca cantidad de muestras y una gran cantidad de genes (hasta decenas de miles). Por ello, es importante aplicar métodos pertinentes para la reducción de la dimensión de los datos y la selección de genes relevantes y no redundantes en la clasificación de las muestras. Existen tres tipos de selección de variables que han sido aplicados en datos de *microarrays* [25]. El primer tipo consiste en técnicas de filtrado (*filter*), donde se seleccionan aquellas variables que correlacionan en mayor medida con las clases según algún test estadístico como t-test o F-test sin tener en cuenta ningún modelo de clasificación. Aunque este método es robusto y rápido computacionalmente presenta algunas limitaciones como la selección de variables redundantes o la pérdida de interacciones no lineales entre las variables. El segundo tipo comprende los métodos *wrapper* donde se van seleccionando pequeños grupos de variables de los datos originales y con ellos se alimenta un algoritmo de clasificación dado hasta encontrar la combinación de variables con mayor rendimiento en la clasificación de las muestras. Sin embargo, estos métodos son muy costosos computacionalmente y presentan problemas de *over-fitting*. El tercer tipo incluye los métodos *embedded* en los que se alimenta el predictor con todas las variables y se utiliza el modelo generado para ordenar las variables según la

importancia en el predictor utilizado.

Los métodos *embedded* han sido muy utilizados en la selección de genes en datos de microarrays para resolver problemas de clasificación de muestras. Entre ellos destacan los algoritmos denominados *recursive feature elimination* (RFE) [26]. En estos, un clasificador es alimentado con todas las variables y se ordena su importancia según el modelo generado. Aquellas variables menos importantes son eliminadas y se alimenta de nuevo el clasificador con las variables restantes. Este paso se repite recurrentemente hasta obtener las variables que permiten conseguir el mayor rendimiento en la clasificación. En [27] utilizan esta aproximación acoplada a un clasificador SVM para identificar los genes más relevantes en la clasificación de muestras procedentes de pacientes con cáncer. En [28] utilizan una variante del método SVM-RFE para lidiar con la clasificación de muestras pertenecientes a múltiples clases de cáncer.

1.1.2.3 Reducción de la dimensión de datos de microarrays con *machine learning*

Otra forma de lidiar con la gran dimensión de los datos es obtener la representación de los datos originales en un espacio de dimensión reducida para su posterior uso en los predictores de clasificación. Un ejemplo clásico es el análisis de componentes principales (PCA, del inglés *Principal component analysis*) que transforma los datos ortogonalmente en componentes lineales no correlacionados entre ellos [29]. Sin embargo, el PCA no es capaz de lidiar con relaciones no lineales entre las variables. En este ámbito se han explorado distintas técnicas sobre datos de expresión génica como el *kernel* PCA (KPCA) o más recientemente los *autoencoders* [30,31]. El *autoencoder* consiste en una arquitectura de red neuronal donde los datos con la dimensión original se hacen pasar por una primera fase de codificación con una o más capas de la red hasta una capa intermedia con una dimensión menor que representa las características de los datos originales. Entonces, los datos representados en la capa intermedia pasan por las capas de la red de manera simétrica a la de codificación y se vuelven a reconstituir los datos con la dimensión original de los mismos. La capa intermedia con la representación latente de los datos en una dimensión reducida puede ser utilizada para alimentar algoritmos en problemas de clasificación. En los últimos años distintas variantes de este algoritmo han sido propuestas para el análisis de datos de expresión génica [32–34].

1.1.3 Justificación del trabajo

A pesar del gran potencial de las técnicas de *machine learning* su uso en el análisis de expresión génica de especies vegetales ha sido poco explorado hasta el momento [11,35]. Uno de los motivos es el poco consenso en la anotación de los ensayos de *microarrays* en los repositorios públicos. Este hecho dificulta el uso de una gran cantidad de muestras ya que se hace necesario descargar individualmente los archivos crudos de expresión y anotar manualmente la información requerida para la clasificación de las muestras (por ejemplo, el tejido del que proviene la muestra). Otro de los problemas es la gran variabilidad en las condiciones

de crecimiento de las plantas usadas en los distintos ensayos. Cada laboratorio utiliza sus propios protocolos donde las plantas son crecidas en medios de cultivo o en tierra, en distintas condiciones de fotoperiodo, intensidad de luz, temperatura, etc. Además de las condiciones propias utilizadas para el objetivo de cada estudio, por ejemplo, tratamiento de salinidad, infección con agentes bióticos o tratamiento con distintas hormonas. Esta variabilidad hace necesario la pre-selección adecuada de genes según el problema de clasificación a solucionar.

En este trabajo se pretende aprovechar la gran cantidad de datos de microarrays pertenecientes a la especie *Arabidopsis thaliana* para generar una base de datos anotada con perfiles de expresión pertenecientes a distintas estructuras representativas del ciclo de vida de la planta. Con estos datos se propone evaluar el potencial de las técnicas de *machine learning* para clasificar perfiles de expresión según el tejido o estructura de la que provienen y descubrir nuevos genes implicados en el desarrollo de las plantas con una aproximación aún no explorada en la bibliografía para este fin.

1.2 Objetivos del Trabajo

1. Evaluar técnicas de machine learning para la clasificación en las distintas estructuras (clases) que definen el ciclo de vida de Arabidopsis thaliana a partir de datos de expresión génica.
    - Aplicar distintos algoritmos de machine learning para la clasificación multi-clase a partir de datos crudos de expresión génica.
    - Comparar el rendimiento de los modelos generados.
2. Definir los genes con mayor relevancia en la identificación de las estructuras estudiadas a partir de los modelos de clasificación obtenidos.
    - Extraer los genes más importantes en la clasificación de las muestras.
    - Analizar la función biológica *in silico* de los genes seleccionados.

1.3 Materiales y métodos

1.3.1 Obtención de los datos

La base de datos se generó a partir de ensayos de microarrays disponibles en el repositorio ArrayExpress (EMBL-EBI). Se utilizaron perfiles de expresión obtenidos en la plataforma de microarray de Affymetrix con mayor cobertura del genoma de Arabidopsis thaliana, Affymetrix GeneChip Arabidopsis Genome [ATH1-121501]. Para disminuir el ruido en los posteriores análisis no se utilizaron réplicas de una misma muestra. Además, para disiminuir la variabilidad en las muestras biólogicas utilizadas, solamente se utilizaron muestras del ecotipo salvaje Col-0 correspondientes a las siguientes estructuras:

- Semilla (*seed*).
- Plántulas (*seedling*) de 3 a 8 días de edad.
- Raíz (*root*) de plantas adultas de 18 a 35 días de edad.

- Hoja (*leaf*) de plantas adultas 18 a 35 días de edad.
- Inflorescencias o flores aisladas (*flower*).

Los ensayos descargados se anotaron manualmente según la información disponible en las publicaciones asociadas.

Los archivos crudos con extensión .CEL fueron procesados y analizados con la función `mas()` del paquete `affy` de Bioconductor. Esta función implementa el método de la *suite* MAS 5.0 de *Affymetrix* para la obtención de los valores de expresión de las sondas. La base de datos generada consiste en 576 perfiles de expresión génica de las distintas estructuras representativas del ciclo de vida de *Arabidopsis thaliana*.

1.3.2 Normalización por cuantil

La normalización por cuantil tiene como objetivo igualar la distribución de la expresión génica provinentes de arrays obtenidos en distintos experimentos para poder compararlos entre sí. Se utilizó la función `normalize.quantiles()` del paquete `preprocessCore` basado en el método propuesto por [36]. Esta normalización consiste en la transformación de los datos con distintas distribuciones en una distribución común igualando los cuantiles de la distribución de la siguiente manera:

- Para un set de datos X de dimensión p x n, donde p representan las sondas y n los ensayos de microarrays;
- Se ordenan descendentemente los valores de las columnas de X, obteniendo X_{sort};
- Se obtiene la media de las filas de X_{sort} y se asigna su valor a cada elemento de la fila obteniéndose X'_{sort};
- Se obtiene $X_{normalizado}$ reordenado las columnas de X'_{sort} con el mismo orden que los datos originales X.

1.3.3 Filtro bivariante

Para la eliminación de genes no relevantes se aplicó un filtro bivariante sobre los datos, a fin de seleccionar sólamente aquellos genes cuya expresión esté relacionada con las clases de las muestras. Para ello se creó una función personalizada en R `filter squared()` que sigue los siguientes pasos:

- Para un set de datos X de dimensión i x j ($X[i,j]$), donde i son las muestras de microarrays y j los genes, y un vector `class` de dimensión igual a i con las etiquetas de las clases a las que pertenecen las muestras.
- Se aplica la función `lm()` de R sobre cada columna de X y se genera una regresión con `X[,j]` y la variable categórica `class`.
- Se obtiene el coeficiente de determinación R^2 asociado a cada recta de regresión y se asigna al gen correspondiente.
- Se seleccionan aquellos genes con $R^2 > corr$, donde $corr$ es un umbral determinado por el usuario. En este caso se utilizó $corr = 0.45$

1.3.4 Presencia/ausencia de transcritos

Para evaluar la representación de un transcrito según esté presente (P), marginal (M) o ausente (A), se utilizó la función `mas5calls()` del paquete `affy`. Esta función implementa el método *detection calls* de MAS 5.0 [37]. Este método consiste en evaluar si el nivel de expresión de un determinado transcrito está por encima del umbral de detección. Si se considera ausente significa que probablemente este nivel no sea distinto de cero. Para ello, se evaluan las señales crudas correspondientes a un transcrito en las que existen dos tipos de sondas: *perfect match* (PM) y *mismatch* (MM). Para asignar la presencia o ausencia de transcrito se siguen cuatro pasos:

- Eliminación de sondas donde: PM ~ MM + *tau*, donde *tau* es un valor predefinido.
- Cálculo del puntaje de discriminación [R]. R es una medida relativa de la diferencia de las intensidades entre PM y MM. Para un transcrito dado i, R se calcula:

  $$R_i = \frac{PM_i - MM_i}{PM_i + MM_i}$$

  Si la mediana(R_i)>τ, se puede rechazar que PM = MM. Se utilizó el valor por defecto, $\tau = 0.015$.
- Para calcular el grado de confianza del valor R y asignar un p-value a este resultado se utiliza el test no paramétrico prueba de los rangos con signo de Wilcoxon, como alternativa a la prueba t de Student.
- Asignar P, A o M según el p-value obtenido. Por defecto se considera: P si p-value $<$ 0.04, M si 0.04 $<$ p-value $<$ 0.06, A si p-value $>$ 0.06.

1.3.5 Partición de los datos

El set de datos resultante del filtrado con 576 muestras y 3652 genes se dividió aleatoriamente en dos grupos denominados *train* y *test*. El grupo *train* contiene el 80% de las muestras y fue utilizado para la aplicación de los algoritmos de selección de genes y para entrenar posteriormente los algoritmos de clasificación con los genes seleccionados. El grupo *test* se utilizó solamente para evaluar el rendimiento de los distintos modelos de clasificación sobre los genes seleccionados.

1.3.6 Métodos de remuestreo

1.3.6.1 *Bootstrapping*

El *bootstrapping* consiste en una técnica de remuestreo donde en cada iteracción se utiliza un n número de las muestras originales sobre la que se aplica el modelo y el resto de muestras sirven para evaluar el rendimiento del modelo. En este caso utilizó el remuestreo por *bootstrapping* en el algoritmo de selección de genes FPRF con 5 iteracciones. En cada iteracción el muestreo se da con reemplazamiento.

1.3.6.2 *Cross-validation*

El método de *cross-validation* (*cv*) consiste en una técnida de remuestreo sin reemplazamiento en que los datos originales se dividen en k grupos de muestras. En cada iteracción 1 de los k grupos se utiliza para evaluar el modelo entrenado sobre los otros $k-1$ grupos. Se utilizó *k-fold cv* durante todos los algoritmos de clasificación y selección de genes por *machine learning*, excepto en el algoritmo FPRF.

1.3.7 Algoritmos de selección de genes con *machine learning*

1.3.7.1 Fuzzy pattern - random forest (FP-RF)

El algoritmo FP-RF fue propuesto en [38] para la selección de genes en problemas de clasificación con múltiples clases. El primer paso, FP, consiste en la implementación de un método de discretización de variables disponible en el paquete de R `DFP` que tiene como objetivo seleccionar un grupo de variables independientes y relevantes en una o más clases [39]. Para ello se aplican dos pasos: i) una aproximación polinomial Gaussiana para evaluar la expresión normal de un gen dado, ii) una aproximación polinomial sigmoide para evaluar los valores extremos y asignar niveles "bajos" o "altos" a un gen según su expresión. Como resultado se aplican etiquetas lingüísticas según el nivel de expresión ("alto", "medio", "bajo"). Finalmente, se seleccionan aquellos genes que presentan frecuentemente etiquetas de "alto" nivel de expresión en al menos una de las clases. En este proceso se utilizan dos parámetros *zeta* y *piVal*. El parámetro *zeta* determina el umbral para determinar los valores de expresión extremos y la asignación de las etiquetas lingüísticas. El valor *piVal* determina el porcentaje de valores "altos" de un gen en una determinada clase para ser seleccionado. En este caso se establecieron valores para estos parámetros que permitiera seleccionar un número alto de variables relevantes ($zeta = 0.5$, $piVal = 0.7$).

El segundo paso de este algoritmo, RF, consiste en la implementación de un algoritmo de clasificación por random forest con la función `ctree` del paquete `party` de R. Para alimentar el algoritmo se utilizó la lista de variables FP seleccionadas en el paso anterior y se evaluó su importancia en el clasificador según un test de permutación. Se seleccionaron grupos de los 30, 50, 100, 150, 200, 250 o 300 genes más importantes y se evaluó su capacidad de clasificación por random forest.

El algoritmo completo se aplicó durante 5 iteracciones por *bootstrapping* y se obtuvo la media y la desviación estándar del valor *accuracy* obtenido en la clasificación con cada grupo de variables.

1.3.7.2 Recursive feature elimination

La aplicación del algoritmo RFE se realizó con la función `rfe` del paquete `caret` de R [40]. El algoritmo consiste en los siguientes pasos:

- Se ajusta un modelo de clasificación con todas las variables predictoras.
- Se calcula el rendimiento del modelo con todas las variables.

- Se calcula la importancia de las variables en el modelo.
- Para cada elemento S_i, $i = 1...S$, donde S es el vector con el tamaño de los grupos, se hace:
  - Se conservan las S_i variables más importantes.
  - Se entrena el modelo de clasificación usando S_i variables.
  - Se calcula el rendimiento del modelo.
  - Se recalcula la importancia de las variables predictoras.
- Se calcula el rendimiento medio de cada grupo de variables S_i.
- Se determina el número de variables con mayor rendimiento.

El procedimiento entero del algoritmo se realizó con un remuestreo por *cross-validation* con *5-fold cv*. Finalmente, se calculó la media y la desviación estándar del *accuracy* del modelo durante las 5 iteracciones del algoritmo RFE en cada grupo de variables S_i.

El vector S que contiene el tamaño de las variables a retener en cada paso fue $S = 3000, 2000, 1500, 1000, 750, 500, 250, 200, 150, 100, 80, 60, 40, 30, 20, 10, 5$. El algoritmo RFE se aplicó utilizando SVM o RF como modelos de clasificación con *10-fold cv*.

1.3.7.3 Autoencoder

El algoritmo de autoencoder se aplicó con el paquete `keras` de R utilizando como *backend* el lenguaje *TensorFlow* desarrollado por Google [41]. El modelo del autoencoder sigue el siguiente esquema:

- Capa entrada, nodos = dimensión de los datos originales.
- Capa intermedia 1, nodos = 400, activación ReLU, tasa de *dropout* = 0.1.
- Capa intermedia 2, nodos = 50, activación ReLU. Capa con la representación latente de los datos.
- Capa intermedia 3, nodos = 400, activación ReLU, tasa de *dropout* = 0.1.
- Capa de salida, nodos = dimensión de capa de entrada, activación sigmoide. Capa con los datos reconstruidos.

Para la aplicación del autoencoder se dividió el grupo de datos *train* en un nuevo grupo *train.ae* con el 70% de los datos y un grupo *test.ae* en el que se evaluó el valor de *accuracy* en la reconstrucción de los datos. Se aplicó el algoritmo durante 500 iteracciones (*epochs*).

La representación latente de los datos con una dimensión de 50 variables se utilizó para la evaluación posterior de la clasificación de las muestras.

1.3.8 Algoritmos de clasificación con *machine learning*

1.3.8.1 Support vector machines

El algoritmo SVM se aplicó con un *kernel* lineal con la función `train` del paquete `caret` de R. Se utilizó como método de remuestreo *5-fold cv* en los problemas de clasificación final de los genes seleccionados y *10-fold cv* durante el algoritmo de RFE. En cada caso se calculó la

media y la desviación estándar de las medidas de rendimiento pertinentes a lo largo de cada iteracción.

1.3.8.2 Random forest

El algoritmo RF se aplicó con un *kernel* lineal con la función `train` del paquete `caret` de R. Se utilizó como método de remuestreo *5-fold cv* en los problemas de clasificación final de los genes seleccionados y *10-fold cv* durante el algoritmo de RFE. El número de árboles se estableción en 1000 en todos los casos. En cada caso se calculó la media y la desviación estándar de las medidas de rendimiento pertinentes a lo largo de cada iteracción.

1.3.8.3 Redes neuronales artificiales

El algoritmo de ANN se implementó con la función `train` del paquete `caret`. En la clasificación con todos los genes seleccionados tras el filtrado de los datos se aplicó una red neuronal de una capa con un nodo y un valor de *decay* de 0.1. En la clasificación de los datos con los genes seleccionados con los distintos algoritmos de *machine learning* se utilizó una red con una capa y tres nodos y un valor de *decay* de 0.1. En todos los casos se utilizó *3-fold cv*. En cada caso se calculó la media y la desviación estándar de las medidas de rendimiento pertinentes a lo largo de cada iteracción.

1.3.9 Métricas de los algoritmos de *machine learning*

- **Matriz de confusión**. Es una matriz que permite evaluar el rendimiento de un algoritmo de *machine learnig*. Las columnas representan las predicciones de cada clase y las filas representan las etiquetas reales de cada clase. En este caso las matrices de confusión generadas tienen el aspecto de la tabla 1.

Tabla 1: Matriz de confusión.

Reales \ Predichos	**Flower**	**Leaf**	**Root**	**Seed**	**Seedling**	**TP+FP**
Flower	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$	$x_{1,5}$	$\sum_{j=1}^{5} x_{(1,j)}$
Leaf	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$	$x_{2,5}$	$\sum_{j=1}^{5} x_{(2,j)}$
Root	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$	$x_{3,5}$	$\sum_{j=1}^{5} x_{(3,j)}$
Seed	$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$	$x_{4,5}$	$\sum_{j=1}^{5} x_{(4,j)}$
Seedling	$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$	$x_{5,5}$	$\sum_{j=1}^{5} x_{(5,j)}$
TP+FN	$\sum_{i=1}^{5} x_{(i,1)}$	$\sum_{i=1}^{5} x_{(i,2)}$	$\sum_{i=1}^{5} x_{(i,3)}$	$\sum_{i=1}^{5} x_{(i,4)}$	$\sum_{i=1}^{5} x_{(i,5)}$	**Total**

Al tratarse de un problema de clasificación con múltiples clases se ha de tener en cuenta los valores para cada una de las clases:

- TP = Elementos de la diagonal.
- TP + TN: Suma de la diagonal = Todos los elementos clasificados correctamente
- TP + FP : Suma de columnas. Número predicho para cada clase.

- TP + FN : Suma de filas. Número real de muestras para cada clase.

  Donde TP = verdaderos positivos, TN = verdaderos negativos, FN = falsos positivos, FP = falsos positivos.

- **Accuracy**. Mide el número de elementos clasificados correctamente y se define como:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

  El *accuracy* total se calcula como el promedio del *accuracy* de todas las clase.

- **F-score**. El F-score está basado en el $F - measure$, que se define como la media armónica que combina los valores *recall* y precisión:

$$F_i - measure = \frac{2R_iP_i}{R_i + P_i}; \forall i = 1 : k$$

  Donde R es el valor *recall*, definido como:

$$Recall = \frac{TP}{TP + FN}$$

  P el valor de precisión definido como:

$$Precisión = \frac{TP}{TP + FP}$$

  E i es la i clase del total de clases k. El F-score total se calcula como el promedio de los $F - measure$ de todas las clases.

- **G-mean**. El valor G-mean se define como la media geométrica de la precisión a lo largo de todas las clases:

$$G - mean = \left(\prod_{i=1}^{k} R_i\right)^{1/k}$$

Tanto el F-score como el valor G-mean son métricas adecuadas para tratar problemas de clasificación en múltiples clases sobretodo cuando las clases no están balanceadas en el número de muestras [42].

1.3.10 Evaluación de la función biológica de los genes seleccionados

1.3.10.1 Anotación génica

Se utilizó la anotación disponible en la versión TAIR10 del genoma de *Arabidopsis thaliana* disponible en la web *The Arabidopsis Information Resource (TAIR) (https://www.arabidopsis.org/).

1.3.10.2 Enriquecimiento de ontologías génicas

El análisis de enriquecimiento de ontologías génicas se realizó en la plataforma VirtualPlant [43]. Se obtuvo el p-value asociado a las categorías enriquecidas con un test de Fisher con correción por FDR.

1.4 Planificación del Trabajo

1.4.1 Tareas

1. Obtención de los datos

   1.1 Descarga y anotación de las muestras de microarrays

   1.2 Lectura, análisis y transformación de los datos

2. Aplicación de algoritmos de clasificación

   2.1 Entrenamiento y evaluación de algoritmos de machine learning

   2.2 Comparación del rendimiento de los modelos de clasificación

3. Identificación de genes

   3.1 Extracción de los genes más importantes en la clasificación de las muestras

   3.2 Anotación génica y análisis ontológico de los genes seleccionados

4. Defensa del trabajo de fin de máster

   4.1 Redacción y entrega de la memoria

   4.2 Elaboración de la presentación

1.4.2 Calendario

Tabla 2: Temporización de las tareas.

Tareas	Inicio	Fin	Duración
Obtención de los datos	**oct 12**	**oct 24**	**13d**
Descarga y anotación de las muestras de microarrays	oct 12	oct 18	7d
Lectura, análisis y transformación de los datos	oct 18	oct 24	7d
Aplicación de algoritmos de clasificación	**oct 25**	**nov 25**	**32d**
Entrenamiento y evaluación de algoritmos de machine learning	oct 25	nov 18	25d
Comparación del rendimiento de los modelos de clasificación	nov 19	nov 25	7d
Identificación de genes	**nov 26**	**dic 12**	**17d**
Extracción de los genes más importantes en la clasificación de las muestras	nov 26	dic 5	10d
Anotación génica y análisis ontológico de los genes seleccionados	dic 6	dic 12	7d
Defensa del trabajo de fin de máster	**dic 1**	**ene 9**	**37 d**
Redacción y entrega de la memoria	dic 1	dic 30	30d
Elaboración de la presentación	ene 3	ene 9	7d

1.5 Breve sumario de productos obtenidos

- **Memoria del TFM.** Se utilizó Rmarkdown para elaborar la memoria del TFM y se generaró un documento PDF a partir de éste como entregable.
- **Código.** Archivo en formato .R con el código utilizado para generar los resultados del trabajo.
- **Base de datos con archivos .CEL de muestras anotadas.** En el siguiente enlace de google drive se encuentran disponibles todos los archivos crudos utilizados y la información de los mismos https://drive.google.com/drive/folders/12jC2SdYd_EhGqO_ZGMH3QBRvCgEqWQBK?usp=sharing
- **Presentación virtual.**
- **Autoevaluación del proyecto.**

1.6 Breve descripción de los otros capítulos de la memoria

- **Resultados**. Este capítulo consiste en la descripción de los resultados obtenidos junto con las figuras y tablas generadas en el estudio.
- **Conclusiones**. Enumeración de las conclusiones extraídas del trabajo. Además contiene la evaluación la consecución de los objetivos propuestos, el seguimiento de la planificación y las perspectivas futuras derivadas del trabajo elaborado.
- **Glosario**
- **Bibliografía**. Enumeración de las referencias utilizadas en el trabajo.
- **Anexos**. Código generado y tablas suplementarias.

Capítulo 2

Resultados

2.1 Obtención y procesado de los datos

2.1.1 Obtención y análisis de los datos

Para este trabajo se generó una base de datos anotada manualmente de ensayos de microarrays obtenidos del repositorio público ArrayExpress (EMBL-EBI). Los datos consisten en 576 microarrays con los perfiles de expresión génica correspondientes a cinco tipos de estructuras representativas del ciclo de vida de la especie *Arabidopsis thaliana* (Fig. 1). Cada perfil de expresión contiene la información de la expresión de 22810 sondas que representa aproximadamente el 80% del genoma de *A. thaliana*.

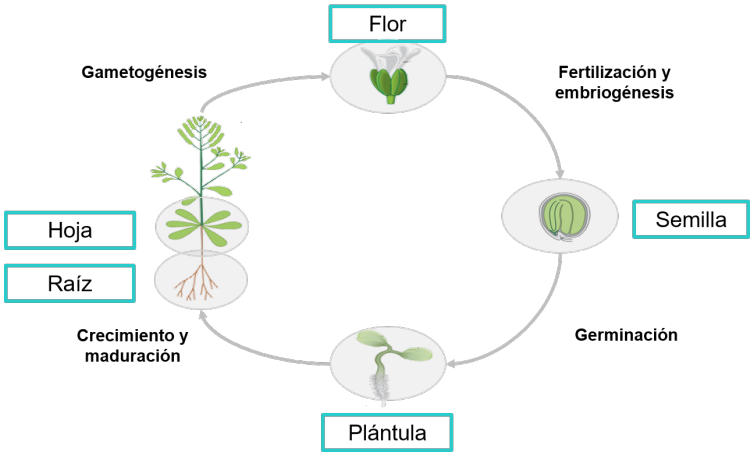

Figura 1: Ciclo de vida de *Arabidopsis thaliana*. En condiciones favorables, las semillas germinan y dan lugar a plántulas que se desarrolla y maduran en la fase vegetativa en la que aumenta el tamaño de la roseta y las raíces. Tras la transición floral se desarrollan las flores que contienen el ovario y los estambres y en las que se da la fertilización y posterior embriogénesis dando lugar a nuevas semillas. Los círculos grises engloban la representación del tejido utilizado como clase en este trabajo y los recuadros turquesas contienen las etiquetas de dichas clases. Adaptado de *www.mun.ca/biology/desmid/brian/BIOL3530/DEVO_07/devo_07.html*

La tabla 3 muestra la frecuencia absoluta de cada una de las clases en la base de datos.

Tabla 3: Frecuencia absoluta de las clases en la base de datos.

Clase	Frecuencia
flower	56
leaf	193
root	78
seed	47
seedling	202

Los datos de microarrays de cDNA de distintas muestras contienen variaciones no asociadas con la información biológica si no con el proceso de obtención de los datos *per se*. Por lo tanto, para evaluar las diferencias de expresión génicas asociadas al origen biológico y no a las características del experimento se han de normalizar los datos antes de su análisis.

Para ello, se aplicó la normalización por cuantil, donde se transforman los datos de expresión de las distintas muestras con distintas distribuciones para que tengan una distribución común y sean comparables entre sí [36]. En la figura 2 se observa la distribución de la señal de expresión de todas las sondas antes y después de la normalización de los datos en quince experimentos de microarrays seleccionados aleatoriamente.

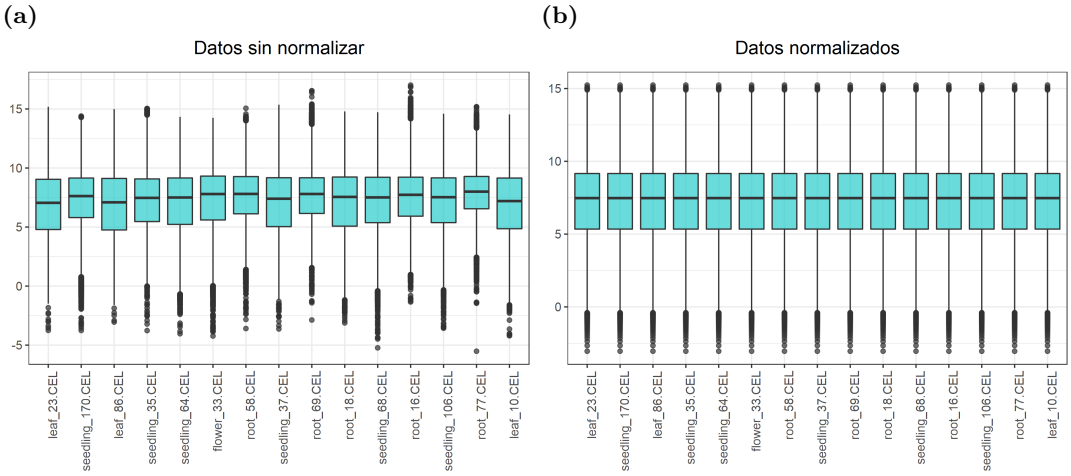

Figura 2: Distribución de los valores de expresión. Los diagramas de caja muestran la distribución de la expresión de los datos crudos **(a)** sin normalizar y **(b)** tras la normalización por cuantil. Cada caja representa la distribución de la señal de todas las sondas en un *microarray*.

Para evaluar la estructura multivariante de los datos se realizó un análisis de componentes principales sobre los datos de expresión normalizados. Las cinco primeras componentes están asociadas con alrededor del 47% de la variabilidad de los datos (Fig. 3a). Las muestras utilizadas en este estudio provienen de plantas crecidas en condiciones muy diversas asociadas al tipo de experimento para el que fueron utilizadas (por ejemplo, plantas sometidas a tratamientos químicos, estreses, etc.) y a los protocolos y equipamientos de cada laboratorio

(intensidad de luz, plantas crecidas en tierra o *in vitro*, temperatura, fotoperiodo, etc.). A pesar de todas estas variables, las muestras se agrupan respondiendo a las características morfológicas de la estructura utilizada (Fig. 3b y 3c). La primera componente podría estar asociada a la presencia de estructuras verdes en las muestras, ya que separa mayoritariamente las clases *flower* y *leaf* de las clases *root* y *seed*. Por su parte, la segunda componente separa las clases *seed* y *flower*, correspondientes a la fase reproductiva de la planta, de las clases *root*, *seedling* y *leaf*, correspondientes a estructuras representativas de la fase vegetativa. En conjunto, este análisis muestra que existe una clara asociación entre los datos de expresión génica y el tipo de estructura de la planta.

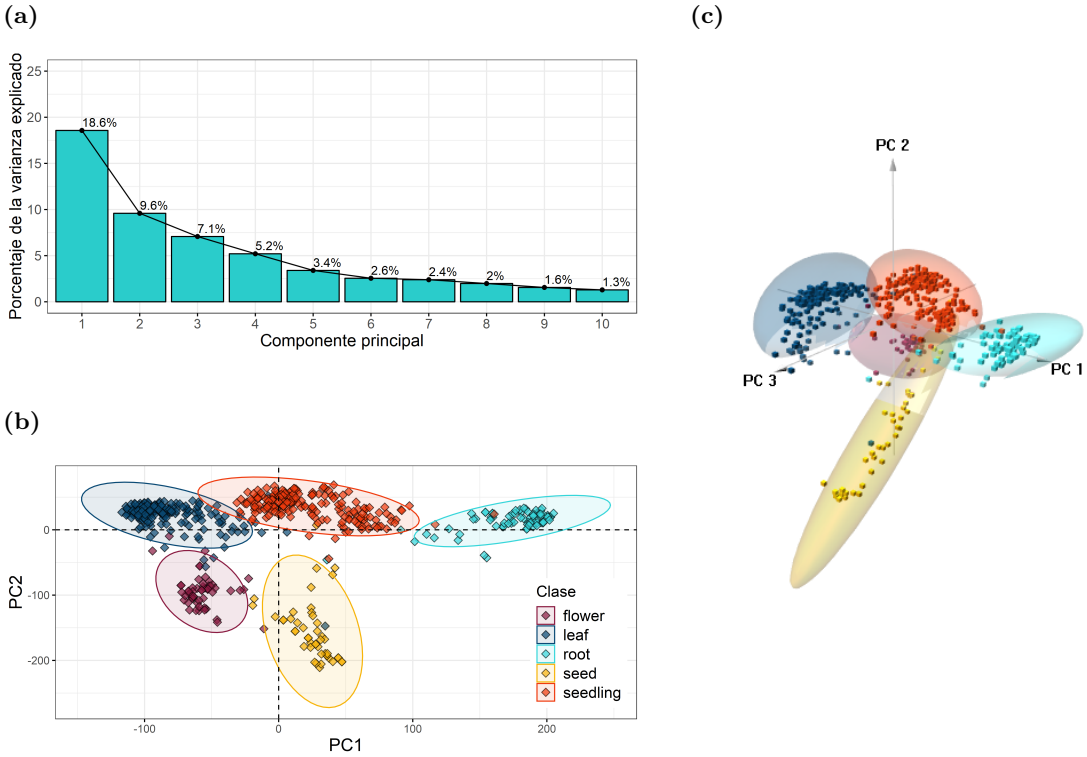

Figura 3: Análisis de componentes principales de los datos de expresión normalizados. **(a)** Scree plot de las primeras 10 componentes principales que muestra el porcentaje de la varianza explicado por cada una. **(b)** y **(c)** PCA de las dos primeras componentes y las tres primeras componentes sobre las 576 muestras. Las muestras están coloreadas según la clase a la que pertenecen y englobadas en la elipse de concentración correspondiente.

2.1.2 Pre-selección de variables

Para la aplicación de los algoritmos de ML es importante realizar una pre-selección de variables con el fin de reducir la dimensión de los datos [44]. De esta forma se consigue i) eliminar los genes irrelevantes en la clasificación de las muestras y ii) mejorar el rendimiento en la aplicación de los algoritmos de ML reduciendo los requerimientos computacionales.

En primer lugar, se evaluó la ausencia o presencia del transcrito según los niveles de expresión en las distintas clases. El número de genes detectados en las muestras es similar en todas las clases y va desde el 55% al 68% de los genes (Fig. 4). Estos datos son consistentes con los reportados anteriormente en la bibliografía [3]. Con el fin de eliminar los genes con baja expresión a lo largo de las muestras se desecharon áquellos que no estuvieran presentes en al menos 47 muestras, que representa la frecuencia absoluta de la clase menos representada *seed*. Tras este primer paso de pre-selección se eliminaron del estudio 3971 genes.

En segundo lugar, se estimó un modelo lineal entre la expresión de cada gen a lo largo de todas las muestras según la clase a la que pertenecen. Se calculó el coeficiente de determinación asociado a cada una de las rectas de regresión estimadas y se seleccionaron sólo los genes cuyo coeficiente asociado fuera mayor a 0.45. Tras este filtro bivariante se pre-seleccionaron 3652 genes.

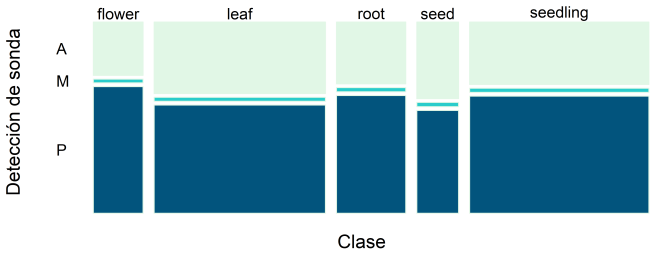

Figura 4: Frecuencia del nivel de detección de transcrito por clase. Gráfico mosaico que representa la tabla de contingencia entre la detección de sondas y las clases. Los rectángulos están coloreados según la presencia (P), ausencia (A) o detección marginal (M) de las sondas. El tamaño de los rectángulos es proporcional al tamaño de los grupos.

La estructura multivariante de los datos procesados fue evaluada por un análisis de componentes principales (Fig. 5). La estructura de los datos alrededor de las tres primeras componentes tiene una estructura similar a la de los datos originales, pero la variabilidad explicada por estas componentes aumentó hasta el 63%.

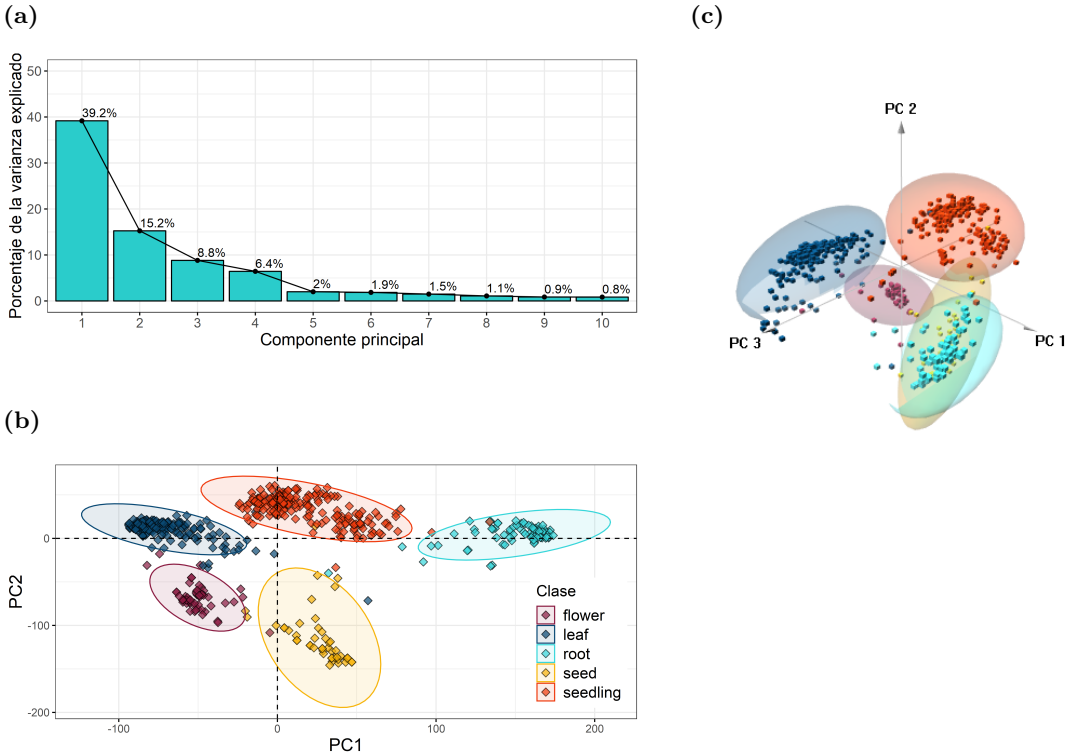

Figura 5: Análisis de componentes principales de los datos de expresión tras el filtrado.(a) Scree plot de las primeras 10 componentes principales que muestra el porcentaje de la varianza explicado por cada una. **(b)** y **(c)** PCA de las dos primeras componentes y las tres primeras componentes sobre las 576 muestras. Las muestras están coloreadas según la clase a la que pertenecen y englobadas en la elipse de concentración correspondiente.

2.2 Selección de genes con *machine learning*

Uno de los objetivos de este trabajo es conocer que genes determinan las distintas estructuras durante el desarrollo de *A. thaliana* según sus patrones de expresión. La selección de genes clave para la clasificación de las muestras por *machine learning* en sus distintos tejidos provee una herramienta potente para la identificación de genes putativamente implicados en el desarrollo de la planta. Para la consecución de este objetivo se siguió el esquema general presentado en la figura 6.

Los datos procesados, que contienen 576 muestras y 3652 genes, se dividieron de manera aleatoria en dos grupos de datos. El primero, denominado *train*, contiene 460 muestras (80% del total) y el segundo, denominado *test*, contiene 116 muestras (20%).

Para la selección de variables se aplicaron distintos algoritmos de *machine learning* sobre el grupo de datos *train*:

- ***Fuzzy pattern - Random Forest* (FP-RF)**

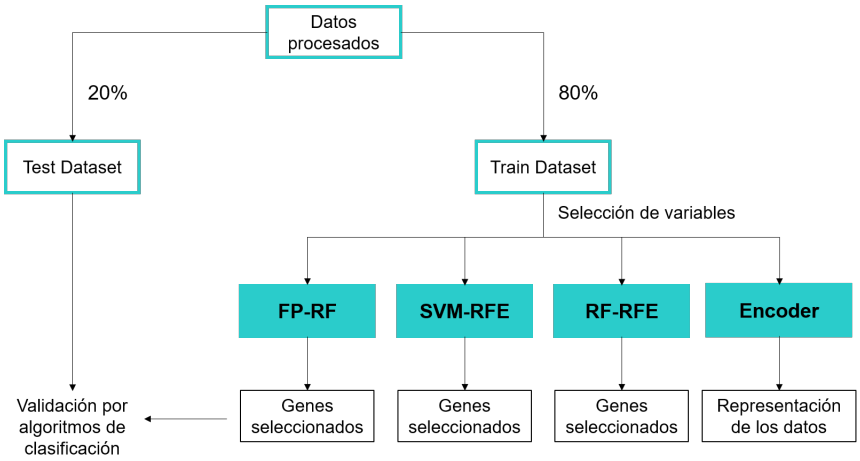

Figura 6: Esquema de trabajo utilizado para la selección de variables.

- ***Recursive feature selection* (RFE)**
  - Basado en *support vector machine* (SVM)
  - Basado en *random forest* (RF)

Este grupo de algoritmos se basa en la selección de variables según su importancia en la clasificación de las muestras de manera supervisada. En todos ellos, se evaluó el valor *accuracy* en la clasificación de las muestras utilizando grupos de variables de distinto tamaño y se seleccionó el número de genes óptimo en cada uno de los algoritmos.

Adicionalmente, se utilió un método no supervisado basado en redes neuronales denominado ***encoder*** que permitió la extracción de un número reducido de variables representativas de los datos originales.

2.2.1 Fuzzy pattern - random forest

El primer método utilizado fue descrito en [38] y consiste en dos pasos principales (Fig. 7). En una primera instancia, se aplica un método de descubrimiento de patrones en el que se asigna una etiqueta nominal ("High", "Medium", "Low") según el valor de expresión de las variables a lo largo de las distintas clases. Después, se seleccionan áquellas variables que son más frecuentemente asignadas como "High" en al menos una de las clases. De esta manera se consigue obtener un set de datos que contiene características independientes y relevantes a las clases. En segundo lugar, se alimenta un algoritmo de RF con estas variables y se asigna la importancia de las mismas en el clasificador con un test de permutación. Las variables se ordenan según su importancia y se seleccionan grupos de distintos tamaños que contienen los genes más relevantes para evaluar el valor de *accuracy* en la clasificación por RF.

Se aplicó el algoritmo FP-RF sobre el grupo de datos *train* que contiene el 80% de las muestras. A su vez este grupo se dividió en dos, un nuevo grupo *train* sobre el que se seleccionaron y priorizaron los genes más relevantes y un grupo *test* en el que se evaluó la

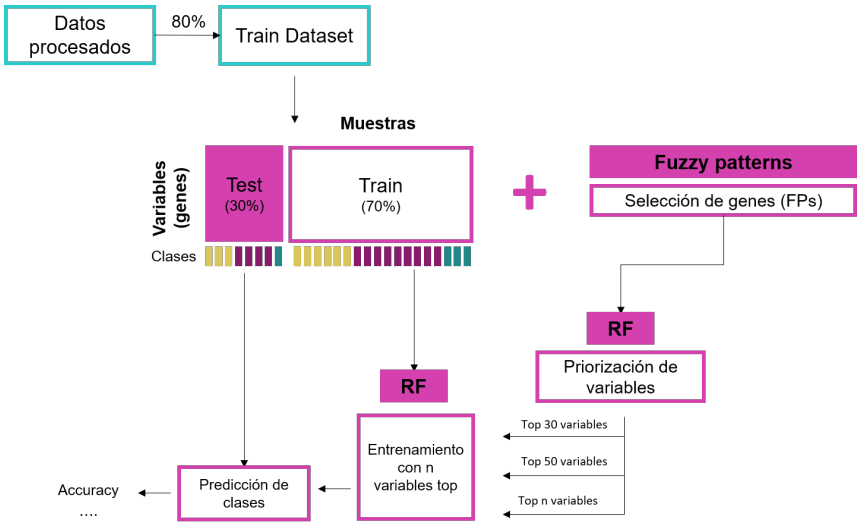

Figura 7: Esquema del algoritmo FP-RF.

capacidad de clasificación de estos genes. Este procedimiento se llevó a cabo en 5 iteraciones por *bootstrapping*. Finalmente, se calculó el *accuracy* medio obtenido a lo largo de las 5 iteraciones utilizando los 30, 50, 100, 150, 200, 250 o 300 genes más relevantes como variables para la clasificación de las muestras por RF (Tabla 4). El mayor valor de *accuracy* se obtuvo utilizando los 30 genes más relevantes.

Tabla 4: Evaluación del rendimiento del algoritmo FP-RF. Media y desviación estándard del valor de *accuracy* a lo largo de 5 iteraciones por *bootstrapping*. Los mejores resultados están resaltados en negrita.

Variables	**Todas**	**30**	**50**	**100**	**150**	**200**	**250**	**300**
Accuracy	0.986 ± 0.013	$\mathbf{0.990 \pm 0.011}$	0.990 ± 0.011	0.984 ± 0.014	0.986 ± 0.014	0.981 ± 0.011	0.983 ± 0.012	0.983 ± 0.012

2.2.2 Recursive feature elimination

El segundo método utilizado para la selección de variables consiste en un proceso recursivo en el que se evalúa la importancia de las variables en un clasificador dado, como puede ser el uso de SVM o RF (Fig. 8). Las variables se ordenan según esta importancia y se elimina la menos relevante (o un grupo de ellas). Con este nuevo grupo de variables se vuelve a alimentar el algoritmo de clasificación y se recalcula su importancia repitiendo el proceso hasta obtener el set de variables óptimo donde el valor *accuracy* de la clasificación es máxima.

Este método se aplicó con el paquete `caret` sobre los datos *train* con *5-fold crossvalidation* con dos tipos de clasificadores, SVM lineal y RF. En cada iteración se utilizó un *10-fold crossvalidation* en el paso de clasificación y predicción de clases. Por el costo computacional del proceso se decidió dejar fuera de cada iteración un número predefinido de variables resultando en cada paso la evaluación del siguiente número de variables: 3652, 3000, 2000, 1500, 1000, 750, 500, 250, 200, 150, 100, 80, 60, 40, 30, 20, 10 y 5.

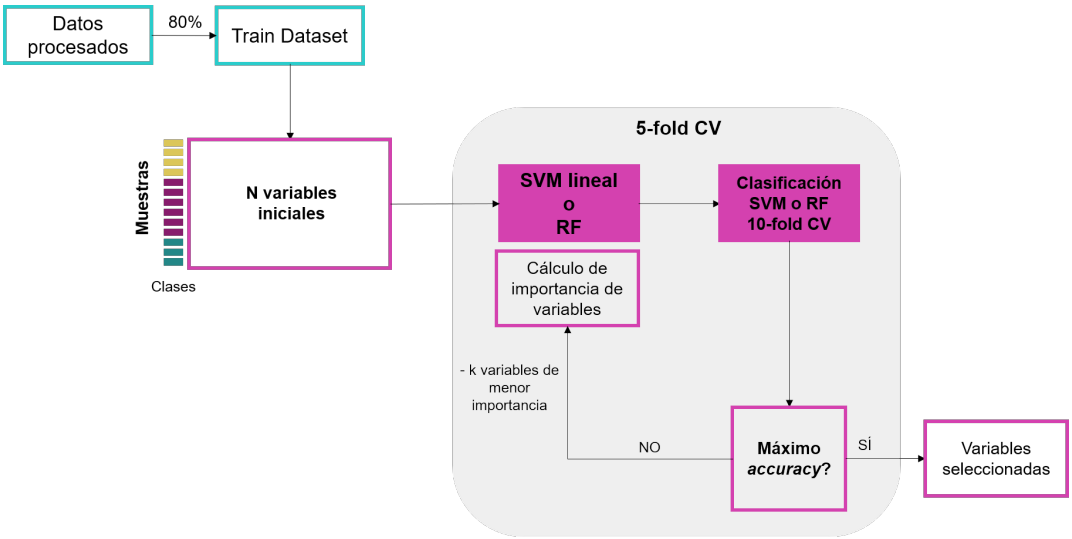

Figura 8: Esquema del algoritmo RFE.

2.2.2.1 *Support Vector Machine - Recursive Feature Elimination*

En el método SVM-RFE se obtuvo el mejor valor de *accuracy* utilizando los 100 genes con mayor peso en el modelo de clasificación, si bien con 10 o 20 genes ya se obtienen valores muy altos de *accuracy* (Tabla 5).

Tabla 5: Evaluación del rendimiento del algoritmo SVM-RFE. Media y desviación estándard del valor de *accuracy* a lo largo de 5 iteraciones por *cross-validation*. Los mejores resultados están resaltados en negrita.

Variables	**Todas**	**5**	**10**	**20**	**30**	**40**	**60**	**80**	**100**
Accuracy	0.989 ± 0.008	0.887 ± 0.104	0.954 ± 0.015	0.985 ± 0.06	0.980 ± 0.016	0.980 ± 0.009	0.980 ± 0.009	0.985 ± 0.006	$\mathbf{0.989 \pm 0.000}$
Variables	**150**	**200**	**250**	**500**	**750**	**1000**	**1500**	**2000**	**3000**
Accuracy	0.987 ± 0.005	0.987 ± 0.005	0.987 ± 0.005	0.989 ± 0.005	0.989 ± 0.008	0.989 ± 0.008	0.989 ± 0.008	0.989 ± 0.008	0.989 ± 0.008

2.2.2.2 *Random Forest - Recursive Feature Elimination*

Con el método RF-RFE se seleccionaron los 20 genes más importantes según el valor de *accuracy* obtenido en la clasificación de las muestras (Tabla 6).

Tabla 6: Evaluación del rendimiento del algoritmo RF-RFE. Media y desviación estándard del valor de *accuracy* a lo largo de 5 iteraciones por *cross-validation*. Los mejores resultados están resaltados en negrita.

Variables	**Todas**	**5**	**10**	**20**	**30**	**40**	**60**	**80**	**100**
Accuracy	0.987 ± 0.009	0.974 ± 0.012	0.980 ± 0.014	$\mathbf{0.991 \pm 0.05}$	0.985 ± 0.010	0.989 ± 0.008	0.989 ± 0.008	0.989 ± 0.008	0.987 ± 0.009
Variables	**150**	**200**	**250**	**500**	**750**	**1000**	**1500**	**2000**	**3000**
Accuracy	0.987 ± 0.009	0.991 ± 0.009	0.991 ± 0.009	0.989 ± 0.008	0.987 ± 0.009	0.985 ± 0.010	0.985 ± 0.010	0.987 ± 0.009	0.987 ± 0.009

2.2.3 Autoencoder

El algoritmo *autoencoder* permite extraer la representación de los datos originales en una dimensión reducida. Utilizando los pesos asociados a los nodos de cada capa también es

posible extraer qué variables son más importantes en esta abstracción, no obstante este proceso no es trivial y requiere de un conocimiento profundo del algoritmo [34,45]. Por ello, en este trabajo solamente se evaluó la capacidad de las variables de la capa latente de representación de los datos en la posterior clasificación de las muestras por técnicas de *machine learning*.

Para obtener la representación latente de los datos se utilizó una red neuronal artificial según el esquema de la figura 9a. La capa latente de dimensión reducida contiene 50 variables. El modelo generado se aplicó a los datos *train* y a los datos *test* para su posterior entrenamiento y evaluación en la clasificación de las muestras.

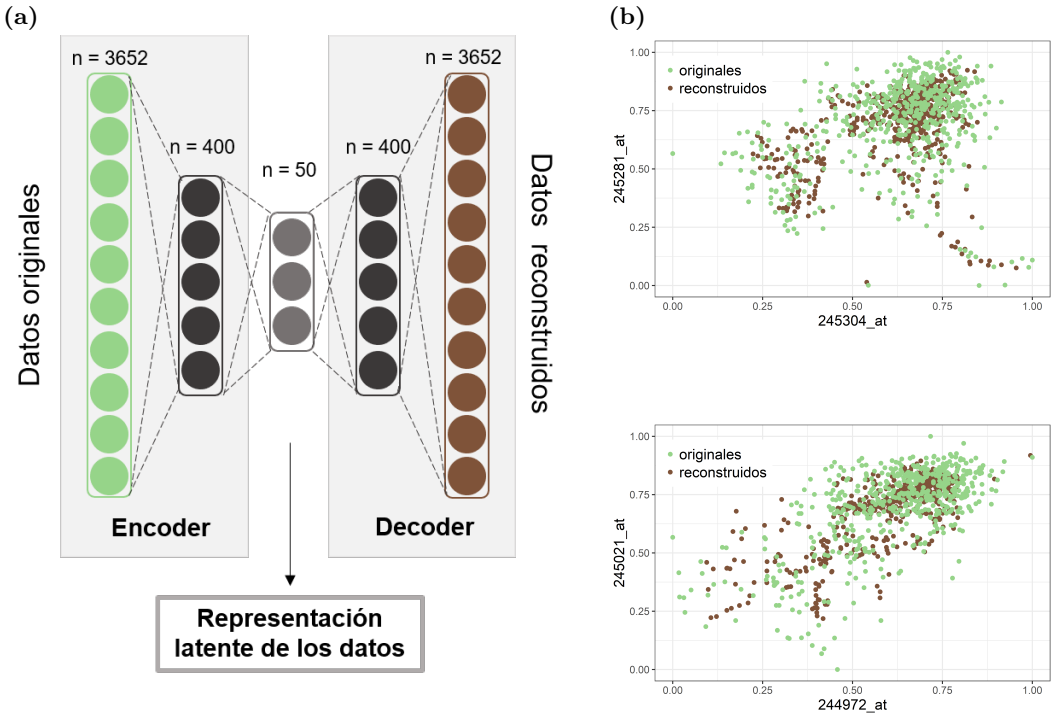

Figura 9: Aplicación del algoritmo *autoencoder*. **(a)** Esquema de la red neuronal del *autoencoder*. Los datos originales se comprimen en un primer paso denominado *encoder* en la que se obtiene una representación latente de los datos en una dimensión reducida. A partir de ésta se generan nuevos datos con la dimensión de los datos originales en la fase denominada *decoder*. El n representa el número de nodos en cada capa de la red. **(b)** Representación de los datos originales y reconstruidos de sondas seleccionadas aleatorimente.

Para evaluar el poder de representación de los datos se obtuvo también la capa de salida de igual dimensión que la capa de datos originales en la que se reconstruyen los datos a partir de la capa latente intermedia. La figura 9b muestra el aspecto de los datos reconstruidos respecto a los datos originales de algunas sondas a lo largo de todas las muestras.

2.3 Clasificación de las muestras con *machine learning*

Para evaluar la potencia de los genes seleccionados en los distintos métodos (o las variables latentes en el caso del *autoencoder*) se utilizaron estos genes para generar distintos modelos de clasificación. Aunque en el paso de selección de variables ya se evaluó la capacidad de predicción de estos genes en un determinado algoritmo, es importante evaluar estos genes como predictores sobre un grupo de datos independiente. En este caso, el grupo de datos *test* nunca fue utilizado en los pasos anteriores para la selección de los genes. Además también es interesante evaluar si los genes seleccionados por un método de clasificación dado, en este caso SVM o RF, presentan un buen rendimiento utilizando otros algoritmos de clasificación para los cuales no fueron entrenados durante la selección.

Se obtuvieron los datos de los grupos *test* y *train* con la expresión de todos los genes (3652), los genes seleccionados por FP-RF (30), SVM-RFE (100), RF-RFE (20) o la representación con el modelo de *autoencoder* (50 variables). Estos datos se utilizaron para generar modelos con distintos clasificadores, SVM lineal, RF o ANN. Los modelos de clasificación SVM lineal y RF se evaluaron con *5-fold cv* sobre los datos *train* y sobre los datos *test*. El modelo basado en ANN se evaluó sobre los datos *train* o *test* normalizados en una escala de 0 a 1 con *3-fold cv*. Se calculó el promedio del valor *accuracy*, *F-score* y *G-mean* a lo largo de todas la iteracciones en cada modelo sobre cada grupo de datos. Los datos obtenidos se muestran en la tabla 7.

Tabla 7: Evaluación del rendimiento de los algoritmos de clasificación. Media y desviación estándar de las métricas de rendimiento de los algoritmos de clasificación entrenados sobre el test de datos *train* o *test* que contienen los genes seleccionados con técnicas de *machine learning*. Los mejores valores de *accuracy* para cada modelo de clasificación están resaltados en negrita.

		SVM		**RF**		**ANN**	
		Train	**Test**	**Train**	**Test**	**Train**	**Test**
Todos	Accuracy	0.98±0.01	0.97±0.01	0.99±0	**1±0**	0.9±0.01	0.87±0.07
	F-score	0.89±0.07	0.85±0.07	0.98±0.01	1±0	0.46±0.03	0.42±0.15
	G-mean	0.94±0.06	0.95±0.01	0.99±0.01	1±0	NaN±NA	NaN±NA
FP-RF	Accuracy	0.89±0.03	0.92±0.04	0.98±0.01	0.98±0.01	0.98±0	0.98±0.02
	F-score	0.68±0.08	0.65±0.08	0.94±0.04	0.89±0.07	0.93±0.02	0.89±0.11
	G-mean	0.74±0.04	NaN±NA	0.95±0.03	0.94±0.05	0.94±0.02	0.95±0.03
SVM-RFE	Accuracy	**0.98±0.01**	**0.99±0.01**	**1±0**	**1±0.01**	0.99±0	**1±0**
	F-score	0.94±0.03	0.96±0.04	0.99±0.01	0.99±0.01	0.99±0.01	1±0
	G-mean	0.95±0.03	0.96±0.04	0.99±0.01	1±0.01	0.98±0.02	1±0
RF-RFE	Accuracy	0.98±0.01	0.93±0.02	**1±0**	**1±0**	**1±0**	**1±0**
	F-score	0.87±0.03	0.69±0.08	0.99±0.02	1±0	0.99±0.01	1±0
	G-mean	0.91±0.03	0.82±NA	0.99±0.01	1±0	0.99±0.01	1±0
Encoder	Accuracy	0.96±0.01	0.98±0.01	0.99±0.01	0.99±0.01	0.99±0.01	0.99±0
	F-score	0.86±0.09	0.9±0.07	0.97±0.04	0.98±0.02	0.98±0.02	0.97±0
	G-mean	0.9±0.04	0.96±0.02	0.97±0.04	0.99±0.01	0.98±0.02	0.97±0.02

En general, se consiguió un buen resultado de clasificación con todas las combinaciones. No obstante, es preferible poder seleccionar un número reducido de genes que tenga por sí solo un alto poder predictivo de las clases lo cual indica que los niveles de expresión de estos grupos

de genes tienen la capacidad de definir la estructura de la planta. El uso como variables predictoras de los 100 genes seleccionados por SVM-RFE presentó los mejores resultados en todos los modelos de clasificación, tanto sobre el grupo de datos *train* como el grupo de datos *test*. También se obtuvieron muy buenos resultados utilizando los 20 genes seleccionados por RF-RFE.

2.4 Análisis funcional de los genes seleccionados

2.4.1 Anotación génica

Los genes seleccionados por *machine learning* demostraron poseer una gran capacidad de clasificación de las muestras en las distintas estructuras de *Arabidopsis thaliana* según sus niveles de expresión. Para entender el posible significado biológico de estos resultados se obtuvo la anotación y descripción de estos genes. En las tablas 8, 9 y 10 se muestra la anotación de los genes seleccionados por FP-RF, RF-RFE y SVM-RFE, respectivamente.

Entre todos los genes seleccionados solamente encontramos uno que es común a los tres métodos y que corresponde al gen RD21B (249187_at). Este gen codifica para una proteína con actividad proteasa que se localiza en la vacuola de las células [46]. Por su parte, los genes ACD6, CRK4, ATSBT4.12, AT4G22212 y AT2G32160 fueron seleccionados tanto con el método FP-RF como por RF-RFE. Los genes AT4G15390, AT2G25980, CER8 y CYP71B7 son compartidos por los métodos SVM-RFE y RF-RFE.

Tabla 8: Anotación de genes seleccionados por FP-RF.

Sonda	AGI	Símbolo	Descripción
249152_s_at	AT5G43350	NA	AT5G43350, phosphate transporter 1
250059_at	AT5G17820	NA	Peroxidase superfamily protein
259009_at	AT3G09260	BGLU23	Glycosyl hydrolase superfamily protein
247091_at	AT5G66390	PRX72	Peroxidase superfamily protein
261930_at	AT1G22440	NA	Zinc-binding alcohol dehydrogenase family protein
246390_at	AT1G77330	ACO5	2-oxoglutarate (2OG) and Fe(II)-dependent oxygenase superfamily protein
246825_at	AT5G26260	NA	TRAF-like family protein
245265_at	AT4G14400	ACD6	ankyrin repeat family protein
246855_at	AT5G26280	NA	TRAF-like family protein
256766_at	AT3G22231	PCC1	pathogen and circadian controlled 1
247297_at	AT5G64100	NA	Peroxidase superfamily protein
254361_at	AT4G22212	NA	Arabidopsis defensin-like protein
259276_at	AT3G01190	NA	Peroxidase superfamily protein
265439_at	AT2G21045	NA	Rhodanese/Cell cycle control phosphatase superfamily protein

Tabla 8: Anotación de genes seleccionados por FP-RF. *(continued)*

Sonda	AGI	Símbolo	Descripción
254392_at	AT4G21600	ENDO5	endonuclease 5
249187_at	AT5G43060	RD21B	Granulin repeat cysteine protease family protein
265698_at	AT2G32160	NA	S-adenosyl-L-methionine-dependent methyltransferases superfamily protein
263153_s_at	AT1G54010	NA	AT1G54010, GDSL-like Lipase/Acylhydrolase superfamily protein
247333_at	AT5G63600	ATFLS5	flavonol synthase 5
254828_at	AT4G12550	AIR1	Auxin-Induced in Root cultures 1
252549_at	AT3G45860	CRK4	cysteine-rich RLK (RECEPTOR-like protein kinase) 4
263437_at	AT2G28670	ESB1	Disease resistance-responsive (dirigent-like protein) family protein
257162_s_at	AT3G24300	NA	AT3G24300, ammonium transporter 1
247755_at	AT5G59090	ATSBT4.12	subtilase 4.12
260130_s_at	AT1G66280	NA	AT1G66280, Glycosyl hydrolase superfamily protein
257673_at	AT3G20370	NA	TRAF-like family protein
259560_at	AT1G21270	WAK2	wall-associated kinase 2
261815_at	AT1G08320	bZIP21	bZIP transcription factor family protein
258080_at	AT3G25930	NA	Adenine nucleotide alpha hydrolases-like superfamily protein
266330_at	AT2G01530	MLP329	MLP-like protein 329

Tabla 9: Anotación de genes seleccionados por RF-RFE.

Sonda	**AGI**	**Símbolo**	**Descripción**
245555_at	AT4G15390	NA	HXXXD-type acyl-transferase family protein
265698_at	AT2G32160	NA	S-adenosyl-L-methionine-dependent methyltransferases superfamily protein
265051_at	AT1G52100	NA	Mannose-binding lectin superfamily protein
265560_at	AT2G05520	ATGRP-3	glycine-rich protein 3
254835_s_at	AT4G12320	NA	AT4G12320, cytochrome P450, family 706, subfamily A, polypeptide 6
247755_at	AT5G59090	ATSBT4.12	subtilase 4.12
254361_at	AT4G22212	NA	Arabidopsis defensin-like protein
257952_at	AT3G21770	NA	Peroxidase superfamily protein
263046_at	AT2G05380	GRP3S	glycine-rich protein 3 short isoform
245265_at	AT4G14400	ACD6	ankyrin repeat family protein

Tabla 9: Anotación de genes seleccionados por RF-RFE. *(continued)*

Sonda	AGI	Símbolo	Descripción
249187_at	AT5G43060	RD21B	Granulin repeat cysteine protease family protein
260531_at	AT2G47240	CER8	AMP-dependent synthetase and ligase family protein
252549_at	AT3G45860	CRK4	cysteine-rich RLK (RECEPTOR-like protein kinase) 4
258897_at	AT3G05730	NA	Encodes a defensin-like (DEFL) family protein.
254130_at	AT4G24540	AGL24	AGAMOUS-like 24
266838_at	AT2G25980	NA	Mannose-binding lectin superfamily protein
249010_at	AT5G44580	NA	unknown protein
259766_at	AT1G64360	NA	unknown protein
266517_at	AT2G35120	NA	Single hybrid motif superfamily protein
262793_at	AT1G13110	CYP71B7	cytochrome P450, family 71 subfamily B, polypeptide 7

Tabla 10: Anotación de genes seleccionados por SVM-RFE.

Sonda	**AGI**	**Símbolo**	**Descripción**
252896_at	AT4G39480	CYP96A9	cytochrome P450, family 96, subfamily A, polypeptide 9
256597_at	AT3G28500	NA	60S acidic ribosomal protein family
258257_at	AT3G26770	NA	NAD(P)-binding Rossmann-fold superfamily protein
262805_at	AT1G20900	AHL27	Predicted AT-hook DNA-binding family protein
264872_at	AT1G24260	AGL9	K-box region and MADS-box transcription factor family protein
264299_s_at	AT1G78850	NA	AT1G78850, D-mannose binding lectin protein with Apple-like carbohydrate-binding domain
245139_at	AT2G45430	AHL22	AT-hook motif nuclear-localized protein 22
259124_at	AT3G02310	AGL4	K-box region and MADS-box transcription factor family protein
262793_at	AT1G13110	CYP71B7	cytochrome P450, family 71 subfamily B, polypeptide 7
264180_at	AT1G02190	NA	Fatty acid hydroxylase superfamily
251826_at	AT3G55110	ABCG18	ABC-2 type transporter family protein
246531_at	AT5G15800	AGL2	K-box region and MADS-box transcription factor family protein
264146_at	AT1G02205	CER1	Fatty acid hydroxylase superfamily

Tabla 10: Anotación de genes seleccionados por SVM-RFE. *(continued)*

Sonda	AGI	Símbolo	Descripción
251987_at	AT3G53280	CYP71B5	cytochrome p450 71b5
259089_at	AT3G04960	NA	Domain of unknown function (DUF3444)
249045_at	AT5G44380	NA	FAD-binding Berberine family protein
266421_at	AT2G38540	ATLTP1	lipid transfer protein 1
266838_at	AT2G25980	NA	Mannose-binding lectin superfamily protein
267460_at	AT2G33810	SPL3	squamosa promoter binding protein-like 3
263738_at	AT1G60060	NA	Serine/threonine-protein kinase WNK (With No Lysine)-related
245328_at	AT4G14465	AHL20	AT-hook motif nuclear-localized protein 20
264057_at	AT2G28550	RAP2.7	related to AP2.7
251979_at	AT3G53140	NA	AT3G53140, O-methyltransferase family protein
254853_at	AT4G12080	AHL1	AT-hook motif nuclear-localized protein 1
261375_at	AT1G53160	FTM6	squamosa promoter binding protein-like 4
251898_at	AT3G54340	AP3	K-box region and MADS-box transcription factor family protein
255768_at	AT1G16705	NA	p300/CBP acetyltransferase-related protein-related
259802_at	AT1G72260	THI2.1	thionin 2.1
256158_at	AT1G13590	ATPSK1	phytosulfokine 1 precursor
264363_at	AT1G03170	FAF2	Protein of unknown function (DUF3049)
263558_at	AT2G16380	NA	Sec14p-like phosphatidylinositol transfer family protein
264830_at	AT1G03710	NA	Cystatin/monellin superfamily protein
264489_at	AT1G27370	SPL10	squamosa promoter binding protein-like 10
264444_at	AT1G27360	SPL11	squamosa promoter-like 11
267639_at	AT2G42200	AtSPL9	squamosa promoter binding protein-like 9
245555_at	AT4G15390	NA	HXXXD-type acyl-transferase family protein
258566_at	AT3G04110	ATGLR1.1	glutamate receptor 1.1
260531_at	AT2G47240	CER8	AMP-dependent synthetase and ligase family protein
248496_at	AT5G50790	AtSWEET10	Nodulin MtN3 family protein
260515_at	AT1G51460	ABCG13	ABC-2 type transporter family protein
245253_at	AT4G15440	CYP74B2	hydroperoxide lyase 1
264708_at	AT1G09740	NA	Adenine nucleotide alpha hydrolases-like superfamily protein
263031_at	AT1G24070	ATCSLA10	cellulose synthase-like A10
259549_at	AT1G35290	ALT1	Thioesterase superfamily protein
249651_at	AT5G37020	ARF8	auxin response factor 8

Tabla 10: Anotación de genes seleccionados por SVM-RFE. *(continued)*

Sonda	AGI	Símbolo	Descripción
258795_at	AT3G04570	AHL19	AT-hook motif nuclear-localized protein 19
249939_at	AT5G22430	NA	Pollen Ole e 1 allergen and extensin family protein
253788_at	AT4G28680	AtTYDC	L-tyrosine decarboxylase
256747_at	AT3G29180	NA	Protein of unknown function (DUF1336)
267144_at	AT2G38110	ATGPAT6	glycerol-3-phosphate acyltransferase 6
247024_at	AT5G66985	NA	unknown protein
265058_s_at	AT1G52040	NA	AT1G52040, myrosinase-binding protein 1
252183_at	AT3G50740	UGT72E1	UDP-glucosyl transferase 72E1
265029_at	AT1G24625	ZFP7	zinc finger protein 7
267425_at	AT2G34810	NA	FAD-binding Berberine family protein
267610_at	AT2G26650	AKT1	K+ transporter 1
260230_at	AT1G74500	ATBS1	activation-tagged BRI1(brassinosteroid-insensitive 1)-suppressor 1
246133_at	AT5G20960	AAO1	aldehyde oxidase 1
249813_at	AT5G23940	DCR	HXXXD-type acyl-transferase family protein
245928_s_at	AT5G24770	NA	AT5G24770, vegetative storage protein 2
256528_at	AT1G66140	ZFP4	zinc finger protein 4
255298_at	AT4G04840	ATMSRB6	methionine sulfoxide reductase B6
246687_at	AT5G33370	NA	GDSL-like Lipase/Acylhydrolase superfamily protein
248684_at	AT5G48485	DIR1	Bifunctional inhibitor/lipid-transfer protein/seed storage 2S albumin superfamily protein
252607_at	AT3G44990	AtXTH31	xyloglucan endo-transglycosylase-related 8
257051_at	AT3G15270	SPL5	squamosa promoter binding protein-like 5
260941_at	AT1G44970	NA	Peroxidase superfamily protein
262827_at	AT1G13100	CYP71B29	cytochrome P450, family 71, subfamily B, polypeptide 29
249835_s_at	AT5G23490	NA	AT5G23490, unknown protein
258617_at	AT3G03000	NA	EF hand calcium-binding protein family
252419_at	AT3G47510	NA	unknown protein
265261_at	AT2G42990	NA	GDSL-like Lipase/Acylhydrolase superfamily protein
258110_at	AT3G14610	CYP72A7	cytochrome P450, family 72, subfamily A, polypeptide 7
263715_at	AT2G20570	ATGLK1	GBF's pro-rich region-interacting factor 1
259327_at	AT3G16460	JAL34	Mannose-binding lectin superfamily protein

Tabla 10: Anotación de genes seleccionados por SVM-RFE. *(continued)*

Sonda	AGI	Símbolo	Descripción
256255_at	AT3G11280	NA	Duplicated homeodomain-like superfamily protein
245418_at	AT4G17370	NA	Oxidoreductase family protein
245554_at	AT4G15380	CYP705A4	cytochrome P450, family 705, subfamily A, polypeptide 4
250722_at	AT5G06190	NA	unknown protein
264137_at	AT1G78960	ATLUP2	lupeol synthase 2
249187_at	AT5G43060	RD21B	Granulin repeat cysteine protease family protein
255644_at	AT4G00870	NA	basic helix-loop-helix (bHLH) DNA-binding superfamily protein
265672_at	AT2G31980	AtCYS2	PHYTOCYSTATIN 2
265132_at	AT1G23830	NA	unknown protein
247696_at	AT5G59780	ATMYB59	myb domain protein 59
262525_at	AT1G17060	CHI2	cytochrome p450 72c1
259966_at	AT1G76500	AHL29	Predicted AT-hook DNA-binding family protein
266322_at	AT2G46690	SAUR32	SAUR-like auxin-responsive protein family
250154_at	AT5G15140	NA	Galactose mutarotase-like superfamily protein
257244_at	AT3G24240	NA	Leucine-rich repeat receptor-like protein kinase family protein
262073_at	AT1G59640	BPE	BIG PETAL P
258675_at	AT3G08770	LTP6	lipid transfer protein 6
259382_s_at	AT3G16430	NA	AT3G16430, jacalin-related lectin 31
250071_at	AT5G18000	VDD	VERDANDI
247553_at	AT5G60910	AGL8	AGAMOUS-like 8
246098_at	AT5G20400	NA	2-oxoglutarate (2OG) and Fe(II)-dependent oxygenase superfamily protein
247107_at	AT5G66040	STR16	sulfurtransferase protein 16
255604_at	AT4G01080	TBL26	TRICHOME BIREFRINGENCE-LIKE 26
245571_at	AT4G14695	NA	Uncharacterised protein family (UPF0041)
267385_at	AT2G44380	NA	Cysteine/Histidine-rich C1 domain family protein

2.4.2 Análisis de expresión de los genes seleccionados

La figura 10 muestra el agrupamiento jerárquico de los genes seleccionados y las muestras según los niveles de expresión génica. Se observa que los grupos de genes seleccionados por los distintos métodos presentan distintos patrones de expresión a lo largo de las muestras. Por ejemplo, entre los genes seleccionados por FP-RF se observa un grupo de genes que

claramente define la clase *leaf* por su altos niveles de expresión y otro grupo de genes que se expresa fuertemente en las clases *root* y *seedling* pero no en *leaf*, *flower* o *seed* (Fig. 10a). Por su parte, la figura 10c muestra los genes seleccionados por SVM-RFE entre los que se encuentran genes específicos de la clase *flower* que se expresan en menor medida a lo largo del resto de clases. Estos resultados sugieren que los distintos métodos resultan en la selección de genes que responden a distintas características biológicas. Además, resulta de gran interés que no todos los genes seleccionados se expresan específicamente en cada una de las clases, si no que su potencia de clasificación corresponde a otras combinaciones de expresión más complejas. Por otro lado, si atendemos al agrupamiento de las muestras según la expresión de los genes seleccionados, se observa que los genes de SVM-RFE son capaces de separar en mayor medida las clases (Fig. 10c). Por su parte, las clases *flower* y *seed* se agrupan conjuntamente teniendo en cuenta los genes seleccionados por FP-RF y RF-RFE (Fig. 10a y 10b).

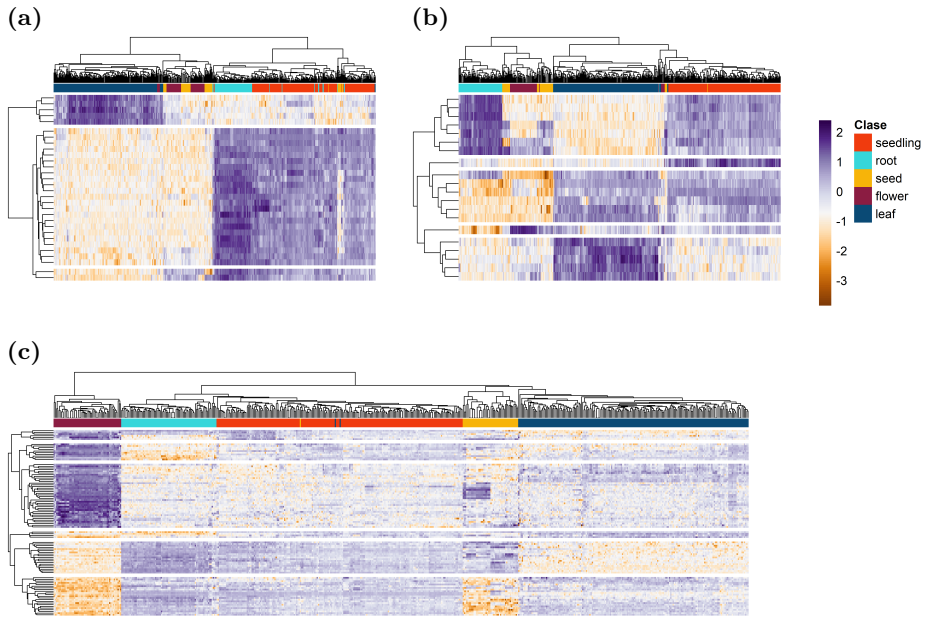

Figura 10: Heatmaps de la expresión de los genes seleccionados. Heatmaps de los genes seleccionados por **(a)** FP-RF **(b)** RF-RFE y **(c)** SVM-RFE. Los heatmaps muestran el nivel de expresión de los genes (filas) a lo largo de todas las muestras (columnas). Los genes y las muestras están agrupados jerarquícamente según la distancia euclidiana entre genes o muestras respectivamente. El gradiente de color naranja-blanco-morado representa niveles de expresión bajos-medios-altos. Las muestras están coloreadas según la clase a la que pertenecen.

2.4.3 Análisis de enriquecimiento de ontologías génicas

Para indagar en la función biológica de los genes seleccionados se hizo un análisis de enriquecimiento ontológico. Los genes seleccionados por el método FP-RF están enriquecidos en categorías relativas a la respuesta a estrés (Fig. 11a). Los genes seleccionados por

RF-RFE sólo están enriquecidos en la ontología de respuesta a ácido salicílico, hormona también relacionada con el estrés biótico (Fig. 11b). Por su parte, los genes seleccionados por SVM-RFE están enriquecidos en ontologías relacionadas con los procesos de desarrollo en general y de desarrollo de algunas estructuras, como las flores o el desarrollo post-embrionario. También aparecen ontologías relacionadas con la síntesis y metabolismo de los ácidos grasos (Fig. 11c).

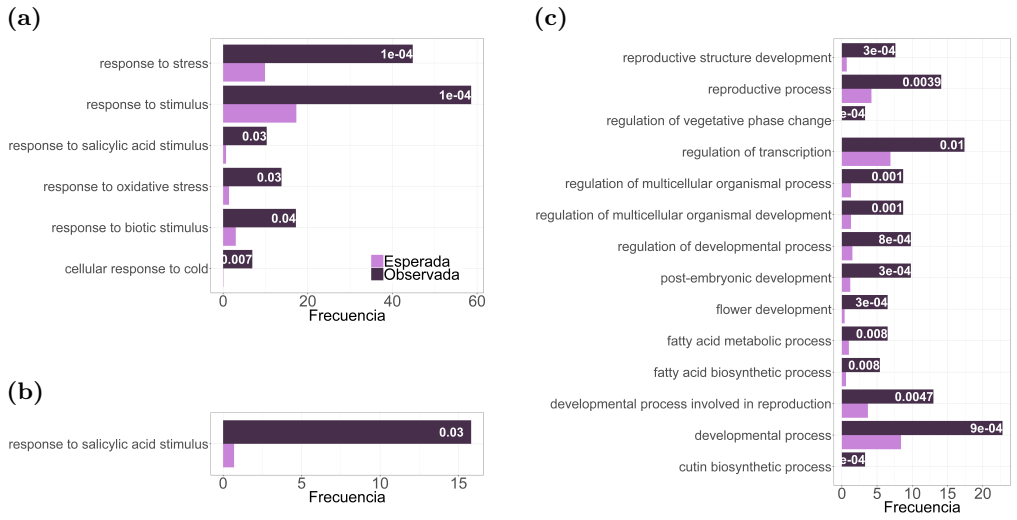

Figura 11: Análisis de enriquecimiento ontológico. Los gráficos de barras muestran el porcentaje de enriquecimiento observado en los genes seleccionados y esperado en el total de los genes presentes en el array ATH1. El p-value asociado al enriquecimiento se muestra en las barras. **(a)** Genes seleccionados por FP-RF. **(b)** Genes seleccionados por RF-RFE. **(c)** Genes seleccionados por SVM-RFE

Estos resultados sugieren que los genes seleccionados por el método SVM-RFE están ligados a los procesos de desarrollo de la planta y puede resultar de gran interés ahondar en su análisis.

Capítulo 3

Conclusiones

En el presente trabajo se generó una base de datos de perfiles de expresión génica, obtenidos por *microarrays*, de las estructuras que definen el ciclo de vida de *Arabidopsis thaliana*. A pesar de la gran variabilidad de las muestras utilizadas, el uso de técnicas de ML permitió generar distintos modelos que consiguen clasificar los patrones de expresión según el tejido de la planta del que provienen con un alto rendimiento.

Además, se exploraron distintos algoritmos de ML para la selección de un número reducido de genes relevantes. Estos genes son suficientes para obtener altos rendimientos de clasificación de las muestras.

Por último, el análisis de la función biológica *in silico* de los genes seleccionados por ML reveló su implicación en el desarrollo de las plantas.

Tras el análisis de estos resultados se ha llegado a las siguientes conclusiones:

- Los algoritmos SVM, RF y ANN generan modelos de alto rendimiento para la clasificación de datos de expresión génica según la estructura de la planta a la que pertenecen.
- Los métodos de selección de variables FP-RF, SVM-RFE y RF-RFE son adecuados para la selección de un número reducido de genes para la clasificación de las muestras de *microarrays*.
- El método *autoencoder* permite la representación de los datos en una dimensión reducida adecuada para la clasificación de las muestras.
- Los genes seleccionados por SVM-RFE consiguen los mejores resultados de clasificación con distintos algoritmos de ML. Además, este grupo de genes está relacionado con procesos de desarrollo en *A. thaliana*.

El conjunto de este trabajo pone de manifiesto la potencia de las herramientas de ML para el descubrimiento de nuevos genes implicados en el desarrollo de las plantas a partir de datos de expresión génica. Sería interesante ahondar en el análisis de estos genes a fin de elucidar

su función biológica en la determinación de las estructuras que definen el ciclo de vida de las plantas. Por ejemplo, se podría determinar cuáles son las redes de expresión asociadas a estos genes en las distintas estructuras. También resulta de interés evaluar la representación de dominios estructurales en las proteínas codificadas por los genes candidatos. Por último, cabe destacar que la base de datos generada y anotada manualmente supone una valiosa herramienta para futuros estudios.

Capítulo 4

Glosario

ANN	Artificial Neural Network
cDNA	ADN complementario
cv	cross-validation
FP	Fuzzy Pattern
ML	Machine learning
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
RF	Random Forest
RFE	Recursive Feature Elimination
SVM	Support Vector machines

Capítulo 5

Bibliografía

[1] D.W. Meinke, J.M. Cherry, C. Dean, S.D. Rounsley, M. Koornneef, Arabidopsis thaliana: A model plant for genome analysis, Science. 282 (1998) 662–682.

[2] P. Huijser, M. Schmid, The control of developmental phase transitions in plants, Development. 138 (2011) 4117–4129.

[3] M. Schmid, T.S. Davison, S.R. Henz, U.J. Pape, M. Demar, M. Vingron, B. Schölkopf, D. Weigel, J.U. Lohmann, A gene expression map of arabidopsis thaliana development, Nature Genetics. 37 (2005) 501.

[4] C.C. Xiang, Y. Chen, CDNA microarray technology and its applications, Biotechnology Advances. 18 (2000) 35–46.

[5] Z. Wang, M. Gerstein, M. Snyder, RNA-seq: A revolutionary tool for transcriptomics, Nature Reviews Genetics. 10 (2009) 57.

[6] U. Nagalakshmi, K. Waern, M. Snyder, RNA-seq: A method for comprehensive transcriptome analysis, Current Protocols in Molecular Biology. 89 (2010) 4–11.

[7] R.A. Irizarry, B.M. Bolstad, F. Collin, L.M. Cope, B. Hobbs, T.P. Speed, Summaries of affymetrix genechip probe level data, Nucleic Acids Research. 31 (2003) e15–e15.

[8] L. Wang, W. Xie, Y. Chen, W. Tang, J. Yang, R. Ye, L. Liu, Y. Lin, C. Xu, J. Xiao, others, A dynamic gene expression atlas covering the entire life cycle of rice, The Plant Journal. 61 (2010) 752–766.

[9] M. Fasoli, S. Dal Santo, S. Zenoni, G.B. Tornielli, L. Farina, A. Zamboni, A. Porceddu, L. Venturini, M. Bicego, V. Murino, others, The grapevine expression atlas reveals a deep transcriptome shift driving the entire plant into a maturation program, The Plant Cell. (2012) tpc–112.

[10] B.H. Le, C. Cheng, A.Q. Bui, J.A. Wagmaister, K.F. Henry, J. Pelletier, L. Kwong, M. Belmonte, R. Kirkbride, S. Horvath, others, Global analysis of gene activity during arabidopsis

seed development and identification of seed-specific transcription factors, Proceedings of the National Academy of Sciences. 107 (2010) 8063–8070.

[11] Y. Ni, D. Aghamirzaie, H. Elmarakeby, E. Collakova, S. Li, R. Grene, L.S. Heath, A machine learning approach to predict gene regulatory networks in seed development in arabidopsis, Frontiers in Plant Science. 7 (2016) 1936.

[12] M. Pirooznia, J.Y. Yang, M.Q. Yang, Y. Deng, A comparative study of different machine learning methods on microarray gene expression data, BMC Genomics. 9 (2008) S13.

[13] K. Kourou, T.P. Exarchos, K.P. Exarchos, M.V. Karamouzis, D.I. Fotiadis, Machine learning applications in cancer prognosis and prediction, Computational and Structural Biotechnology Journal. 13 (2015) 8–17.

[14] A. Statnikov, I. Tsamardinos, Y. Dosbayev, C.F. Aliferis, GEMS: A system for automated cancer diagnosis and biomarker discovery from microarray gene expression data, International Journal of Medical Informatics. 74 (2005) 491–503.

[15] S.B. Kotsiantis, I. Zaharakis, P. Pintelas, Supervised machine learning: A review of classification techniques, Emerging Artificial Intelligence Applications in Computer Engineering. 160 (2007) 3–24.

[16] N.M. Nasrabadi, Pattern recognition and machine learning, Journal of Electronic Imaging. 16 (2007) 049901.

[17] B. Lantz, Machine learning with r, Packt Publishing Ltd, 2013.

[18] C. Cortes, V. Vapnik, Support-vector networks, Machine Learning. 20 (1995) 273–297.

[19] T.S. Furey, N. Cristianini, N. Duffy, D.W. Bednarski, M. Schummer, D. Haussler, Support vector machine classification and validation of cancer tissue samples using microarray expression data, Bioinformatics. 16 (2000) 906–914.

[20] Y. Zhu, X. Shen, W. Pan, Network-based support vector machine for classification of microarray samples, BMC Bioinformatics. 10 (2009) S21.

[21] L. Breiman, Random forests, Machine Learning. 45 (2001) 5–32.

[22] R. Díaz-Uriarte, S.A. De Andres, Gene selection and classification of microarray data using random forest, BMC Bioinformatics. 7 (2006) 3.

[23] B. Liu, Q. Cui, T. Jiang, S. Ma, A combinational feature selection and ensemble neural network method for classification of gene expression data, BMC Bioinformatics. 5 (2004) 136.

[24] J. Khan, J.S. Wei, M. Ringner, L.H. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C.R. Antonescu, C. Peterson, others, Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks, Nature Medicine. 7 (2001) 673.

[25] V. Bolón-Canedo, N. Sánchez-Maroño, A. Alonso-Betanzos, A review of feature selection

methods on synthetic data, Knowledge and Information Systems. 34 (2013) 483–519.

[26] J. Bedo, C. Sanderson, A. Kowalczyk, An efficient alternative to svm based recursive feature elimination with applications in natural language processing and bioinformatics, in: Australasian Joint Conference on Artificial Intelligence, Springer, 2006: pp. 170–180.

[27] I. Guyon, J. Weston, S. Barnhill, V. Vapnik, Gene selection for cancer classification using support vector machines, Machine Learning. 46 (2002) 389–422.

[28] X. Li, X. Gong, X. Peng, S. Peng, SSiCP: A new svm based recursive feature elimination algorithm for multiclass cancer classification, International Journal of Multimedia and Ubiquitous Engineering. 9 (2014) 347–360.

[29] S. Wold, K. Esbensen, P. Geladi, Principal component analysis, Chemometrics and Intelligent Laboratory Systems. 2 (1987) 37–52.

[30] J. Yang, A.F. Frangi, J.-y. Yang, D. Zhang, Z. Jin, KPCA plus lda: A complete kernel fisher discriminant framework for feature extraction and recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence. 27 (2005) 230–244.

[31] X. Lu, Y. Tsao, S. Matsuda, C. Hori, Speech enhancement based on deep denoising autoencoder., in: Interspeech, 2013: pp. 436–440.

[32] B.L. Betechuoh, T. Marwala, T. Tettey, Autoencoder networks for hiv classification, Current Science. (2006) 1467–1473.

[33] R. Fakoor, F. Ladhak, A. Nazi, M. Huber, Using deep learning to enhance cancer diagnosis and classification, in: Proceedings of the International Conference on Machine Learning, ACM New York, USA, 2013.

[34] J. Tan, J.H. Hammond, D.A. Hogan, C.S. Greene, ADAGE-based integration of publicly available pseudomonas aeruginosa gene expression data with denoising autoencoders illuminates microbe-host interactions, MSystems. 1 (2016) e00025–15.

[35] C. Ma, M. Xin, K.A. Feldmann, X. Wang, Machine learning–based differential network analysis: A study of stress-responsive transcriptomes in arabidopsis, The Plant Cell. (2014) tpc–113.

[36] B.M. Bolstad, R.A. Irizarry, M. Åstrand, T.P. Speed, A comparison of normalization methods for high density oligonucleotide array data based on variance and bias, Bioinformatics. 19 (2003) 185–193.

[37] W.-m. Liu, R. Mei, X. Di, T.B. Ryder, E. Hubbell, S. Dee, T.A. Webster, C. Harrington, M.-h. Ho, J. Baid, others, Analysis of high density expression microarrays with signed-rank call algorithms, Bioinformatics. 18 (2002) 1593–1599.

[38] V. Fortino, P. Kinaret, N. Fyhrquist, H. Alenius, D. Greco, A robust and accurate method for feature selection and prioritization from multi-class omics data, PloS One. 9

(2014) e107801.

[39] F. Díaz, F. Fdez-Riverola, D. Glez-Peña, J.M. Corchado, Using fuzzy patterns for gene selection and data reduction on microarray data, in: International Conference on Intelligent Data Engineering and Automated Learning, Springer, 2006: pp. 1087–1094.

[40] M. Kuhn, others, Building predictive models in r using the caret package, Journal of Statistical Software. 28 (2008) 1–26.

[41] T.B. Arnold, KerasR: R interface to the keras deep learning library, The Journal of Open Source Software. 2 (2017).

[42] H. Yu, S. Hong, X. Yang, J. Ni, Y. Dan, B. Qin, Recognition of multiple imbalanced cancer types based on dna microarray data using ensemble classifiers, BioMed Research International. 2013 (2013).

[43] M.S. Katari, S.D. Nowicki, F.F. Aceituno, D. Nero, J. Kelfer, L.P. Thompson, J.M. Cabello, R.S. Davidson, A.P. Goldberg, D.E. Shasha, others, VirtualPlant: A software platform to support systems biology research, Plant Physiology. 152 (2010) 500–515.

[44] Y. Perez-Riverol, M. Kuhn, J.A. Vizcaíno, M.-P. Hitz, E. Audain, Accurate and fast feature selection workflow for high-dimensional omics data, PloS One. 12 (2017) e0189875.

[45] P. Danaee, R. Ghaeini, D.A. Hendrix, A deep learning approach for cancer detection and relevant gene identification, in: PACIFIC Symposium on Biocomputing 2017, World Scientific, 2017: pp. 219–229.

[46] T. Shindo, J.C. Misas-Villamil, A.C. Hörger, J. Song, R.A. van der Hoorn, A role in immunity for arabidopsis cysteine protease rd21, the ortholog of the tomato immune protease c14, PloS One. 7 (2012) e29317.

Capítulo 6

Anexos

```
##Paquetes
###Paquetes CRAN
libraries <- c("ggplot2","knitr","reshape2",
               "gridExtra","ggfortify","devtools",
               "party","caret","randomForest","FactoMineR",
               "factoextra","pca3d")
check.libraries <- is.element(libraries, installed.packages()[, 1])==FALSE
libraries.to.install <- libraries[check.libraries]
if (length(libraries.to.install!=0)) {
  install.packages(libraries.to.install)
}
###Paquetes Bioconductor
success <- sapply(libraries,require, quietly = FALSE,  character.only = TRUE)
if(length(success) != length(libraries))
  {stop("A package failed to return a success in require() function.")}
###Paquetes bioconductor
libraries <- c("affy","ath1121501cdf","ath1121501.db","preprocessCore","DFP")
check.libraries <- is.element(libraries, installed.packages()[, 1])==FALSE
libraries.to.install <- libraries[check.libraries]
if (length(libraries.to.install!=0)) {
  source("https://bioconductor.org/biocLite.R")
  biocLite(libraries.to.install)
}
success <- sapply(libraries,require, quietly = FALSE,  character.only = TRUE)
if(length(success) != length(libraries))
  {stop("A package failed to return a success in require() function.")}

## Directorios
workingDir <-getwd()
sampleDir <- file.path(workingDir, "Muestras")
rawDir <- file.path(workingDir,"RawData")
results <- file.path(workingDir,"Results")

## Extracción de datos a partir de archivos .CEL
```

```
files <- list.files(sampleDir, ".*CEL") #Listado archivos

###Loop para leer intensidades normalizadas y asignar Ausencia/Presencia
all <- list()
for (i in 1:3){
  total <- list()
  x <- ReadAffy(filenames=file.path(sampleDir,files[i])) #affy object
  eset_mas5 <- mas5(x) #ExpressionSet Affymetrix MAS 5.0
  call_mas5 <- mas5calls(x) #Presence/absence detection
  values <- list(exprs(eset_mas5))
  calls <- list(exprs(call_mas5))
  total<-c(values,calls)
  all <- c(all,total)
}

RawTable <- as.data.frame(all)
write.csv(table,file.path(rawDir,"RawTable.csv"))

## Archivo con sólo los valores crudos
Raw_values<- RawTable[,-grep(pattern=".CEL.1",colnames(RawTable))]
write.csv(Raw_values,file.path(rawDir,"RawValues.csv"))

#Archivo con sólo la detección de transcrito
Raw_presence<- RawTable[,grep(pattern=".CEL.1",colnames(RawTable))]
write.csv(Raw_presence,file.path(rawDir,"RawPresence.csv"))

##Normalización por cuantil de los datos
raw_matrix <- as.matrix(Raw_values)
normalized_data<-normalize.quantiles(raw_matrix)
normalized_data<- as.data.frame(normalized_data)
names(normalized_data) <- names(Raw_values)
rownames(normalized_data) <- rownames(Raw_values)
write.csv(normalized_data,file.path(rawDir,"NormalizedData.csv"))
##Transformación logarítimica de los datos normalizados
norm_log_data <- log2(normalized_data)
write.csv(norm_log_data,file.path(rawDir,"NormLogData.csv"))

##Análisis descriptivo de los datos
###Boxplot distribución de los datos normalizados vs no normalizados
set.seed(123)
x <- sample(ncol(Raw_values),15)

ggplot(data=melt(log2(Raw_values[,x])), aes(x=variable, y=value)) +
  geom_boxplot(fill="#2ACCCB",alpha=0.7) +
  theme_bw()+
  theme(axis.title.x=element_blank(),
        axis.title.y=element_blank(),
        axis.text.y = element_text(size=10),
        axis.text.x = element_text(size=10,angle = 90, hjust = 1),
        plot.title = element_text(hjust = 0.5, size=15,
                                  margin = ggplot2::margin(b=12)))+
  ggtitle("Datos sin normalizar")
```

```
ggplot(data=melt(norm_log_data[,x]), aes(x=variable, y=value)) +
  geom_boxplot(fill="#2ACCCB",alpha=0.7) +
  theme_bw()+
  theme(axis.title.x=element_blank(),
        axis.title.y=element_blank(),
        axis.text.y = element_text(size=10),
        axis.text.x = element_text(size=10,angle = 90, hjust = 1),
        plot.title = element_text(hjust = 0.5, size=15,
                                  margin = ggplot2::margin(b=12)))+
  ggtitle("Datos normalizados")


###PCA datos normalizados y transformados logarítmicamente
rel_muestras <- read.csv(file="muestras.csv",
                         header=TRUE,stringsAsFactors = FALSE)
rel_muestras <- rel_muestras[ order(rel_muestras$FileName), ]
#Ordenamos alfábeticamente las muestras en la base de datos
class<-rel_muestras$Class #Extraemos la información referente a las clases
data_t <-as.data.frame(t(norm_log_data)) #Trasponemos los datos
data_plus_class<-cbind(data_t,class) #Añadimos la clase a los datos
pca_analysis<-prcomp(data_plus_class[,-ncol(data_plus_class)]) #pca

screeplot_1 <- fviz_eig(pca_analysis, addlabels = TRUE, ylim = c(0, 25),
                        barfill ="#2ACCCB",barcolor = "black")
ggpubr::ggpar(screeplot_1,
              title = "",
              ggtheme = theme_bw(),
              ylab="Porcentaje de la varianza explicado",
              xlab="Componente principal",
              font.x = 15,
              font.y=15,
              font.tickslab = 12)

pca_plot1 <- fviz_pca_ind(pca_analysis,
                          geom.ind = "point",
                          col.ind = "black", # color by groups
                          palette = c("#8B1C42","#0A4A75",
                                      "#36D6DA","#F7B409","#F03B0E"),
                          pointshape = 23,
                          pointsize = 2.5,
                          fill.ind = data_plus_class$class,
                          addEllipses = TRUE, # Concentration ellipses
                          mean.point =FALSE,
                          alpha.ind = 0.7)

ggpubr::ggpar(pca_plot1,
              title = "",
              legend.title = "Clase",
              legend = c(0.90,0.3),
              ggtheme = theme_bw(),
              font.tickslab = 10,
```

```
              xlab="PC1",
              ylab="PC2",
              font.x = 16,
              font.y=16,
              font.legend = 14)

pca3d(pca_analysis, group=data_plus_class$class,
      palette = c("#8B1C42","#0A4A75",
                  "#36D6DA","#F7B409","#F03B0E"),
      legend=NULL,show.ellipses = TRUE,
      shape= "cube")
## Filtraje de los datos, reducción de la dimensión

###Gráfico mosaico

presence_t <-as.data.frame(t(Raw_presence))
#Trasponemos los datos
presence_plus_class<-cbind(presence_t,class)
#Añadimos la clase a los datos
split_presence <-split(presence_plus_class,presence_plus_class$class)
# Separamos los datos por clase

##Tablas de contigencia detección sonda/clase en frecuencia absoluta
presence_table <- data.frame(rbind(table(unlist(split_presence[[1]]))[1:3],
                                   table(unlist(split_presence[[2]]))[1:3],
                                   table(unlist(split_presence[[3]]))[1:3],
                                   table(unlist(split_presence[[4]]))[1:3],
                                   table(unlist(split_presence[[5]]))[1:3]))
rownames(presence_table) <- c("flower","leaf","root","seed","seedling")

mosaicplot(presence_table,main=NULL,
           color=c("#E1F7E6","#2ACCCB","#02547D"),
           xlab="Clase",
           ylab="Detección de sonda",
           border=c("#E1F7E6","#2ACCCB","#02547D"),
           cex.axis = 1,
           las=par(cex.lab=1.2))

###Eliminación de sondas presentes de expresión ausente.
##Criterio: presencia >47 muestras
presence_filter <- norm_log_data[which(rowSums(RawTable== "P")>46),]
write.csv(presence_filter,file.path(rawDir,"Presence_filter.csv"))

###Filtro por correlación con clases
##Función para la correlación
filter_rsquared <- function(data,class,corr){
  all_coef <-matrix()
  for (i in 1:ncol(data)){
    one_coef <- summary(lm(data[,i]~class))$r.squared
    all_coef[i] <- one_coef
    names(all_coef)[i]<-names(data)[i]
  }
```

```
  attr_subset <- subset(all_coef, all_coef>= corr)
  data <- data[, names(data) %in% names(attr_subset)]
}
##Preparación de los datos
data <- cbind(as.data.frame(t(presence_filter)),class)
##Aplicación del filtro
correlation_filter<-filter_rsquared(data=data[,-ncol(data)],
                                    class = data$class,corr = 0.45)
write.csv(correlation_filter,file.path(rawDir,"Corr_filter.csv"))


#PCA datos filtrados

#Analisis pca
pca_analysis_filt<-prcomp(correlation_filter)
corr_filter_class <- cbind(correlation_filter,class)

#screeplot

screeplot_2 <- fviz_eig(pca_analysis_filt, addlabels = TRUE, ylim = c(0, 50),
                        barfill ="#2ACCCB",barcolor = "black")
ggpubr::ggpar(screeplot_2,
              title = "",
              ggtheme = theme_bw(),
              ylab="Porcentaje de la varianza explicado",
              xlab="Componente principal",
              font.x = 15,
              font.y=15,
              font.tickslab = 12)

pca_plot2 <- fviz_pca_ind(pca_analysis_filt,
                          geom.ind = "point",
                          col.ind = "black",
                          palette = c("#8B1C42","#0A4A75",
                                      "#36D6DA","#F7B409","#F03B0E"),
                          pointshape = 23,
                          pointsize = 2.5,
                          fill.ind = data_plus_class$class,
                          addEllipses = TRUE,
                          mean.point =FALSE,
                          alpha.ind = 0.7)
ggpubr::ggpar(pca_plot2,
              title = "",
              legend.title = "Clase",
              legend = c(0.90,0.3),
              ggtheme = theme_bw(),
              font.tickslab = 10,
              xlab="PC1",
              ylab="PC2",
              font.x = 16,
              font.y=16,
              font.legend = 14)
```

```
pca3d(pca_analysis_filt, group=corr_filter_class$class,
      palette = c("#8B1C42","#0A4A75","#36D6DA","#F7B409","#F03B0E"),
      legend=NULL,show.ellipses = TRUE,
      shape= "cube")

##Parte 2

##Paquetes
###Paquetes CRAN
libraries <- c("ggplot2","knitr","reshape2","gridExtra",
               "devtools","party","caret",
               "randomForest","keras","kerasR",
               "plyr","RColorBrewer","gplots","pheatmap",
               "GOstats")
check.libraries <- is.element(libraries, installed.packages()[, 1])==FALSE
libraries.to.install <- libraries[check.libraries]
if (length(libraries.to.install!=0)) {
  install.packages(libraries.to.install)
}
success <- sapply(libraries,require, quietly = FALSE,  character.only = TRUE)
if(length(success) != length(libraries))
  {stop("A package failed to return a success in require() function.")}
###Paquetes bioconductor
libraries <- c("affy","ath1121501cdf",
               "ath1121501.db","preprocessCore",
               "DFP","clusterProfiler")
check.libraries <- is.element(libraries, installed.packages()[, 1])==FALSE
libraries.to.install <- libraries[check.libraries]
if (length(libraries.to.install!=0)) {
  source("https://bioconductor.org/biocLite.R")
  biocLite(libraries.to.install)
}

success <- sapply(libraries,require, quietly = FALSE,  character.only = TRUE)
if(length(success) != length(libraries))
  {stop("A package failed to return a success in require() function.")}

##Lectura de datos

rel_muestras <- read.csv(file="muestras.csv",
                         header=TRUE,stringsAsFactors = FALSE)
rel_muestras <- rel_muestras[ order(rel_muestras$FileName), ]
#Ordenamos alfábeticamente las muestras en la base de datos
class<-rel_muestras$Class #Extraemos la información referente a las clases
correlation_filter <- read.csv(file.path(rawDir,"Corr_filter.csv"),
                               row.names = 1,check.names = FALSE)

##Set.seed for partition data and classifier

seed.partition <- 12345
seed.classifier <- 54321
```

```
##Train y test data

data<- correlation_filter
set.seed(seed.partition)
n_train <- 0.8
n <- nrow(data)
train <- sample(n,floor(n*n_train))

data.train <- data[train,]
data.test <- data[-train,]
label.train <- class[train]
label.test <- class[-train]

##Algoritmos de machine learning para feature selection

### Selección de variables y clasificación por FPRP

##Algoritmo orginal en Fortino et. al 2014. Ligeramente modificado.

## Funciones para la aplicación del algoritmo

## Feature selection with FP-RP

##Confusion matrix results function
summary.conf.matrix <- function(cm, ...) {

  ## Number of groups
  Ngp <- nrow(cm)

  ## Total : TP + TN + FP + FN
  Tot <- sum(cm)

  ## TP : True positive item : All items on diagonal
  TP <- diag(cm)

  ## TP + TN : sum of diagonal = All correct identification
  TP_TN <- sum(TP)

  ## TP + FP : sum of columns : Automatic classification
  TP_FP <- colSums(cm)

  ## TP + FN : sum of rows : Manual classification
  TP_FN <- rowSums(cm)

  ## FP : False positive items
  FP <- TP_FP - TP

  ## FN : False negative item
  FN <- TP_FN - TP

  ## TN : True Negative = Total - TP - FP - FN
```

```
  TN <- rep(Tot, Ngp) - TP - FP - FN

  ## The 8 basic ratios
  ## Recall = TP / (TP + FN) = 1 - FNR
  Recall <- TP / (TP_FN)

  ## Specificity = TN / (TN + FP) = 1 - FPR
  Specificity <- TN / (TN + FP)

  ## Precision = TP / (TP + FP) = 1 - FDR
  Precision <- TP / (TP_FP)

  ## F-score = F-measure = F1 score = Harmonic mean of Precision and recall
  Fmeasure <- 2 * ((Precision * Recall) / (Precision + Recall))
  Fmeasure[is.nan(Fmeasure)] <- 0
  Fscore <- mean(Fmeasure)
  Gmean <- prod(Recall)^(1/Ngp)

  ## General statistics
  Accuracy <- ((TP + TN) / (TP + TN + FP + FN))
  Gmean.acc <- prod(Accuracy)^(1/Ngp)
  Error <- 1 - Accuracy

  ## Create a data frame with all results
  res <- list(data.frame(Fmeasure = Fmeasure,
                         Recall = Recall,
                         Precision = Precision,
                         Specificity = Specificity,
                         Accuracy,
                         Error),
              Gmean,
              Fscore,
              Gmean.acc)
  return(res)
}

# "filter_data" must be an ExpressionSet onject and the
# corresponding "pheno" field must contain the column "class"
boot.sam.eval <- function(filter_data, nboot = 5, p = 70,
                          zeta = 0.5, piVal = 0.8, ntr=1000) {

  bootSam <- round((table(filter_data$class)/100)*p)
  # percentage of samples to keep
  print("bootSam..")
  print(bootSam)

  nclass  <- names(table(filter_data$class))
  id.class <- list()
  for(i in 1:length(nclass)) {
    print(nclass[i])
    id.class[[i]] <- which(nclass[i] == filter_data$class)
    print(length(id.class[[i]]))
```

```
}

size.fs.fp  <- rep(0, nboot)

cnt.fp.mrf.all <- 0

list.fs.fp.mrf.all <- list()
list.fs.fp.mrf.30  <- list()
list.fs.fp.mrf.50  <- list()
list.fs.fp.mrf.100 <- list()
list.fs.fp.mrf.150 <- list()
list.fs.fp.mrf.200 <- list()
list.fs.fp.mrf.250 <- list()
list.fs.fp.mrf.300 <- list()

fs.freq.fp.mrf.all <- rep(0,dim(exprs(filter_data))[1])
fs.freq.fp.mrf.30  <- rep(0,dim(exprs(filter_data))[1])
fs.freq.fp.mrf.50  <- rep(0,dim(exprs(filter_data))[1])
fs.freq.fp.mrf.100 <- rep(0,dim(exprs(filter_data))[1])
fs.freq.fp.mrf.150 <- rep(0,dim(exprs(filter_data))[1])
fs.freq.fp.mrf.200 <- rep(0,dim(exprs(filter_data))[1])
fs.freq.fp.mrf.250 <- rep(0,dim(exprs(filter_data))[1])
fs.freq.fp.mrf.300 <- rep(0,dim(exprs(filter_data))[1])

## for the binom statistical test
stability.feats <- data.frame(st.pv.fp.mrf.all=rep(0,dim(exprs(filter_data))[1]),
                              st.pv.fp.mrf.30=rep(0,dim(exprs(filter_data))[1]),
                              st.pv.fp.mrf.50=rep(0,dim(exprs(filter_data))[1]),
                              st.pv.fp.mrf.100=rep(0,dim(exprs(filter_data))[1]),
                              st.pv.fp.mrf.150=rep(0,dim(exprs(filter_data))[1]),
                              st.pv.fp.mrf.200=rep(0,dim(exprs(filter_data))[1]),
                              st.pv.fp.mrf.250=rep(0,dim(exprs(filter_data))[1]),
                              st.pv.fp.mrf.300=rep(0,dim(exprs(filter_data))[1]))

## overall ov-acc, f-score, g-mean
acc.fp.mrf.all <- rep(0,nboot)
acc.fp.mrf.30  <- rep(0,nboot)
acc.fp.mrf.50  <- rep(0,nboot)
acc.fp.mrf.100 <- rep(0,nboot)
acc.fp.mrf.150 <- rep(0,nboot)
acc.fp.mrf.200 <- rep(0,nboot)
acc.fp.mrf.250 <- rep(0,nboot)
acc.fp.mrf.300 <- rep(0,nboot)

gmean.fp.mrf.all <- rep(0,nboot)
gmean.fp.mrf.30  <- rep(0,nboot)
gmean.fp.mrf.50  <- rep(0,nboot)
gmean.fp.mrf.100 <- rep(0,nboot)
gmean.fp.mrf.150 <- rep(0,nboot)
gmean.fp.mrf.200 <- rep(0,nboot)
gmean.fp.mrf.250 <- rep(0,nboot)
gmean.fp.mrf.300 <- rep(0,nboot)
```

```
fscore.fp.mrf.all <- rep(0,nboot)
fscore.fp.mrf.30  <- rep(0,nboot)
fscore.fp.mrf.50  <- rep(0,nboot)
fscore.fp.mrf.100 <- rep(0,nboot)
fscore.fp.mrf.150 <- rep(0,nboot)
fscore.fp.mrf.200 <- rep(0,nboot)
fscore.fp.mrf.250 <- rep(0,nboot)
fscore.fp.mrf.300 <- rep(0,nboot)

## Fuzzy Discretization
mfs <- calculateMembershipFunctions(filter_data, skipFactor = 3)
print("calculateMembershipFunctions <DONE>")

for(i in 1:nboot) {

  list.ids.train <- list()
  list.ids.test  <- list()

  #################### SAMPLING FROM EACH CLASS
  for(j in 1:length(nclass)) {
    list.ids.train[[j]] <- sample(id.class[[j]],bootSam[j])
    list.ids.test[[j]]  <- id.class[[j]][which
                                         ((id.class[[j]] %in%
                                             list.ids.train[[j]]) == FALSE)]
  }
  print("Sample size:")
  print(dim(filter_data[,unlist(list.ids.train)]))

  #################### FEATURE SELECTION
  print("Feature selection..")
  time_dvs <-
    system.time(dvs <-
                  discretizeExpressionValues(
                    filter_data[,unlist(list.ids.train)],
                    mfs, zeta = zeta, overlapping=2))
  print("discretizeExpressionValues <DONE>")
  ## Find fuzzy patterns <FPs>
  time_fps <-
    system.time(fps <- calculateFuzzyPatterns(
      filter_data[,unlist(list.ids.train)],
      dvs, piVal, overlapping=2))
  print("calculateFuzzyPatterns <DONE>")
  listFPs <-
    lapply(names(table(
      filter_data[,unlist(list.ids.train)]$class)),
      FUN= function(c) { fp = showFuzzyPatterns(fps, c);
      which( (rownames(filter_data[,unlist(list.ids.train)])
              %in% names(fp)) == TRUE)})
  print("Length of Fuzzy Patterns:")
  print(lapply(listFPs, FUN = length))
```

```
## Store the number of selected features
dfp.features <- rownames(filter_data)[unique(unlist(listFPs))]
size.fs.fp[i]  <- length(dfp.features)

#################### SETTING DATA
print("Datset size <training>: ")
data <-
  data.frame(t(exprs(filter_data[,unlist(list.ids.train)])),
             class = as.factor(filter_data[,unlist(list.ids.train)]$class))
print(dim(data))
print("Datset size <test>: ")
data.test <-
  data.frame(t(exprs(filter_data[,unlist(list.ids.test)])),
             class = as.factor(
               filter_data[,unlist(list.ids.test)]$class))
print(dim(data.test))
print("Datset size <cforest>: ")
map.feats.fp <- data.frame(unlist(listFPs))
list.dec.fp  <- list()
cnt <- 1
for(f in 1:length(nclass)) {
  print(length(listFPs[[f]]))
  list.dec.fp[[f]] <- c(cnt:(length(listFPs[[f]])+cnt-1) )
  cnt <- cnt + length(listFPs[[f]])
}
data_cf  <- data[,c(map.feats.fp[,1], ncol(data))]
data_cf_test  <- data.test[,c(map.feats.fp[,1], ncol(data.test))]
print(dim(data_cf))
print(length(unique(unlist(listFPs))))
print(head(colnames(data)[unique(unlist(listFPs))]))

#################### PRIORITIZATION
print("Classification & Prioritization with RFs..")
### Modified RF based on FPs
time_fprf <-
  system.time(fp.mrf <- cforest(class~.,
                                data = data_cf,
                                control = cforest_control(ntree = ntr)))
# print(time_dvs+time_fps+time_fprf)

#################### COMPUTE IMP. VARIABLES <step>
print("Variable importance..")
fp.mrf.imp <- varimp(fp.mrf)
fp.mrf.imp.wr <- rep(0,length(unique(unlist(listFPs))))
names(fp.mrf.imp.wr) <- colnames(data)[unique(unlist(listFPs))]
for(k in 1:length(fp.mrf.imp.wr)) {
  ids <- which(grepl(names(fp.mrf.imp.wr)[k], names(fp.mrf.imp)) == TRUE)
  fp.mrf.imp.wr[k] <- sum(fp.mrf.imp[ids])
}
print(length(fp.mrf.imp.wr))
```

```
#################### SELECT THE FEATURES TO ASSESS THE POST_SELECTION ACCURACY
list.fs.fp.mrf.all[[i]] <-
  names(fp.mrf.imp.wr[sort(fp.mrf.imp.wr,
                           decreasing = TRUE,
                           index.return = TRUE)[[2]]])
list.fs.fp.mrf.30[[i]]  <-
  na.omit(names(fp.mrf.imp.wr[sort(fp.mrf.imp.wr,
                                   decreasing = TRUE,
                                   index.return = TRUE)[[2]]])[1:30])
list.fs.fp.mrf.50[[i]]  <-
  na.omit(names(fp.mrf.imp.wr[sort(fp.mrf.imp.wr,
                                   decreasing = TRUE,
                                   index.return = TRUE)[[2]]])[1:50])
list.fs.fp.mrf.100[[i]] <-
  na.omit(names(fp.mrf.imp.wr[sort(fp.mrf.imp.wr,
                                   decreasing = TRUE,
                                   index.return = TRUE)[[2]]])[1:100])
list.fs.fp.mrf.150[[i]] <-
  na.omit(names(fp.mrf.imp.wr[sort(fp.mrf.imp.wr,
                                   decreasing = TRUE,
                                   index.return = TRUE)[[2]]])[1:150])
list.fs.fp.mrf.200[[i]] <-
  na.omit(names(fp.mrf.imp.wr[sort(fp.mrf.imp.wr,
                                   decreasing = TRUE,
                                   index.return = TRUE)[[2]]])[1:200])
list.fs.fp.mrf.250[[i]] <-
  na.omit(names(fp.mrf.imp.wr[sort(fp.mrf.imp.wr,
                                   decreasing = TRUE,
                                   index.return = TRUE)[[2]]])[1:250])
list.fs.fp.mrf.300[[i]] <-
  na.omit(names(fp.mrf.imp.wr[sort(fp.mrf.imp.wr,
                                   decreasing = TRUE,
                                   index.return = TRUE)[[2]]])[1:300])

cnt.fp.mrf.all <- cnt.fp.mrf.all + length(list.fs.fp.mrf.all[[i]])

print("Number of selected features..")

ids.fp.mrf.all <- which((colnames(data) %in% list.fs.fp.mrf.all[[i]]) == TRUE)
print(length(ids.fp.mrf.all))
fs.freq.fp.mrf.all[ids.fp.mrf.all] <- fs.freq.fp.mrf.all[ids.fp.mrf.all] + 1

ids.fp.mrf.30 <- which((colnames(data) %in% list.fs.fp.mrf.30[[i]]) == TRUE)
print(length(ids.fp.mrf.30))
fs.freq.fp.mrf.30[ids.fp.mrf.30] <- fs.freq.fp.mrf.30[ids.fp.mrf.30] + 1

ids.fp.mrf.50 <- which((colnames(data) %in% list.fs.fp.mrf.50[[i]]) == TRUE)
print(length(ids.fp.mrf.50))
fs.freq.fp.mrf.50[ids.fp.mrf.50] <- fs.freq.fp.mrf.50[ids.fp.mrf.50] + 1

ids.fp.mrf.100 <- which((colnames(data) %in% list.fs.fp.mrf.100[[i]]) == TRUE)
print(length(ids.fp.mrf.100))
```

```
fs.freq.fp.mrf.100[ids.fp.mrf.100] <- fs.freq.fp.mrf.100[ids.fp.mrf.100] + 1

ids.fp.mrf.150 <- which((colnames(data) %in% list.fs.fp.mrf.150[[i]]) == TRUE)
print(length(ids.fp.mrf.150))
fs.freq.fp.mrf.150[ids.fp.mrf.150] <- fs.freq.fp.mrf.150[ids.fp.mrf.150] + 1

ids.fp.mrf.200 <- which((colnames(data) %in% list.fs.fp.mrf.200[[i]]) == TRUE)
print(length(ids.fp.mrf.200))
fs.freq.fp.mrf.200[ids.fp.mrf.200] <- fs.freq.fp.mrf.200[ids.fp.mrf.200] + 1

ids.fp.mrf.250 <- which((colnames(data) %in% list.fs.fp.mrf.250[[i]]) == TRUE)
print(length(ids.fp.mrf.250))
fs.freq.fp.mrf.250[ids.fp.mrf.250] <- fs.freq.fp.mrf.250[ids.fp.mrf.250] + 1

ids.fp.mrf.300 <- which((colnames(data) %in% list.fs.fp.mrf.300[[i]]) == TRUE)
print(length(ids.fp.mrf.300))
fs.freq.fp.mrf.300[ids.fp.mrf.300] <- fs.freq.fp.mrf.300[ids.fp.mrf.300] + 1

#################### EVALUATE THE ACCURACY
print("Evaluate the accuracy..")

ev.fp.mrf.all <-
  randomForest(x=data[,c(ids.fp.mrf.all,ncol(data))],
               y=data$class,
               xtest=data.test[,c(ids.fp.mrf.all,ncol(data.test))],
               ytest=data.test$class, ntree=ntr)

ev.fp.mrf.30  <-
  randomForest(x=data[,c(ids.fp.mrf.30,ncol(data))],
               y=data$class,
               xtest=data.test[,c(ids.fp.mrf.30,ncol(data.test))],
               ytest=data.test$class, ntree=ntr)

ev.fp.mrf.50  <-
  randomForest(x=data[,c(ids.fp.mrf.50,ncol(data))],
               y=data$class,
               xtest=data.test[,c(ids.fp.mrf.50,ncol(data.test))],
               ytest=data.test$class, ntree=ntr)

ev.fp.mrf.100 <-
  randomForest(x=data[,c(ids.fp.mrf.100,ncol(data))],
               y=data$class,
               xtest=data.test[,c(ids.fp.mrf.100,ncol(data.test))],
               ytest=data.test$class, ntree=ntr)

ev.fp.mrf.150 <-
  randomForest(x=data[,c(ids.fp.mrf.150,ncol(data))],
               y=data$class,
               xtest=data.test[,c(ids.fp.mrf.150,ncol(data.test))],
               ytest=data.test$class, ntree=ntr)

ev.fp.mrf.200 <-
```

```
  randomForest(x=data[,c(ids.fp.mrf.200,ncol(data))],
               y=data$class,
               xtest=data.test[,c(ids.fp.mrf.200,ncol(data.test))],
               ytest=data.test$class, ntree=ntr)

ev.fp.mrf.250 <-
  randomForest(x=data[,c(ids.fp.mrf.250,ncol(data))],
               y=data$class,
               xtest=data.test[,c(ids.fp.mrf.250,ncol(data.test))],
               ytest=data.test$class, ntree=ntr)

ev.fp.mrf.300 <-
  randomForest(x=data[,c(ids.fp.mrf.300,ncol(data))],
               y=data$class,
               xtest=data.test[,c(ids.fp.mrf.300,ncol(data.test))],
               ytest=data.test$class, ntree=ntr)

gmet.ev.fp.mrf.all <-
  summary.conf.matrix(
    ev.fp.mrf.all$confusion[
      ,-ncol(ev.fp.mrf.all$confusion)])
gmet.ev.fp.mrf.30  <-
  summary.conf.matrix(
  ev.fp.mrf.30$confusion[
    ,-ncol(ev.fp.mrf.30$confusion)])
gmet.ev.fp.mrf.50  <- summary.conf.matrix(
  ev.fp.mrf.50$confusion[
    ,-ncol(ev.fp.mrf.50$confusion)])
gmet.ev.fp.mrf.100 <- summary.conf.matrix(
  ev.fp.mrf.100$confusion[
    ,-ncol(ev.fp.mrf.100$confusion)])
gmet.ev.fp.mrf.150 <- summary.conf.matrix(
  ev.fp.mrf.150$confusion[
    ,-ncol(ev.fp.mrf.150$confusion)])
gmet.ev.fp.mrf.200 <- summary.conf.matrix(
  ev.fp.mrf.200$confusion[
    ,-ncol(ev.fp.mrf.200$confusion)])
gmet.ev.fp.mrf.250 <- summary.conf.matrix(
  ev.fp.mrf.250$confusion[
    ,-ncol(ev.fp.mrf.250$confusion)])
gmet.ev.fp.mrf.300 <- summary.conf.matrix(
  ev.fp.mrf.300$confusion[
    ,-ncol(ev.fp.mrf.300$confusion)])

## Get the accuracy
acc.fp.mrf.all[i] <- mean(ev.fp.mrf.all$test$predicted == data.test$class)
acc.fp.mrf.30[i]  <- mean(ev.fp.mrf.30$test$predicted == data.test$class)
acc.fp.mrf.50[i]  <- mean(ev.fp.mrf.50$test$predicted == data.test$class)
acc.fp.mrf.100[i] <- mean(ev.fp.mrf.100$test$predicted == data.test$class)
acc.fp.mrf.150[i] <- mean(ev.fp.mrf.150$test$predicted == data.test$class)
acc.fp.mrf.200[i] <- mean(ev.fp.mrf.200$test$predicted == data.test$class)
acc.fp.mrf.250[i] <- mean(ev.fp.mrf.250$test$predicted == data.test$class)
```

```
  acc.fp.mrf.300[i] <- mean(ev.fp.mrf.300$test$predicted == data.test$class)

  ## Get the G-mean
  gmean.fp.mrf.all[i] <- gmet.ev.fp.mrf.all[[2]]
  gmean.fp.mrf.30[i]  <- gmet.ev.fp.mrf.30[[2]]
  gmean.fp.mrf.50[i]  <- gmet.ev.fp.mrf.50[[2]]
  gmean.fp.mrf.100[i] <- gmet.ev.fp.mrf.100[[2]]
  gmean.fp.mrf.150[i] <- gmet.ev.fp.mrf.150[[2]]
  gmean.fp.mrf.200[i] <- gmet.ev.fp.mrf.200[[2]]
  gmean.fp.mrf.250[i] <- gmet.ev.fp.mrf.250[[2]]
  gmean.fp.mrf.300[i] <- gmet.ev.fp.mrf.300[[2]]

  ## Get the Fscore
  fscore.fp.mrf.all[i] <- gmet.ev.fp.mrf.all[[3]]
  fscore.fp.mrf.30[i]  <- gmet.ev.fp.mrf.30[[3]]
  fscore.fp.mrf.50[i]  <- gmet.ev.fp.mrf.50[[3]]
  fscore.fp.mrf.100[i] <- gmet.ev.fp.mrf.100[[3]]
  fscore.fp.mrf.150[i] <- gmet.ev.fp.mrf.150[[3]]
  fscore.fp.mrf.200[i] <- gmet.ev.fp.mrf.200[[3]]
  fscore.fp.mrf.250[i] <- gmet.ev.fp.mrf.250[[3]]
  fscore.fp.mrf.300[i] <- gmet.ev.fp.mrf.300[[3]]
}

freq.fp.all  <- rep(0,nboot)
for(j in 1:nboot) { freq.fp.all[j] <- length(which(fs.freq.fp.mrf.all == j)) }

freq.mrf.30  <- rep(0,nboot)
for(j in 1:nboot) { freq.mrf.30[j] <- length(which(fs.freq.fp.mrf.30 == j)) }

freq.mrf.50  <- rep(0,nboot)
for(j in 1:nboot) { freq.mrf.50[j] <- length(which(fs.freq.fp.mrf.50 == j)) }

freq.mrf.100 <- rep(0,nboot)
for(j in 1:nboot) { freq.mrf.100[j] <- length(which(fs.freq.fp.mrf.100 == j)) }

freq.mrf.150 <- rep(0,nboot)
for(j in 1:nboot) { freq.mrf.150[j] <- length(which(fs.freq.fp.mrf.150 == j)) }

freq.mrf.200 <- rep(0,nboot)
for(j in 1:nboot) { freq.mrf.200[j] <- length(which(fs.freq.fp.mrf.200 == j)) }

freq.mrf.250 <- rep(0,nboot)
for(j in 1:nboot) { freq.mrf.250[j] <- length(which(fs.freq.fp.mrf.250 == j)) }

freq.mrf.300 <- rep(0,nboot)
for(j in 1:nboot) { freq.mrf.300[j] <- length(which(fs.freq.fp.mrf.300 == j)) }

st.df <- data.frame(freq.fp.all=freq.fp.all,
                    freq.mrf.30=freq.mrf.30,
                    freq.mrf.50=freq.mrf.50,
                    freq.mrf.100=freq.mrf.100,
                    freq.mrf.150=freq.mrf.150,
```

```
                      freq.mrf.200=freq.mrf.200,
                      freq.mrf.250=freq.mrf.250,
                      freq.mrf.300=freq.mrf.300)

  sc.st      <- list(list.fs.fp.mrf.all, list.fs.fp.mrf.30,
                     list.fs.fp.mrf.50, list.fs.fp.mrf.100, list.fs.fp.mrf.150,
                     list.fs.fp.mrf.200, list.fs.fp.mrf.250, list.fs.fp.mrf.300)

  sc.st.freq <- list(fs.freq.fp.mrf.all, fs.freq.fp.mrf.30,
                     fs.freq.fp.mrf.50, fs.freq.fp.mrf.100, fs.freq.fp.mrf.150,
                     fs.freq.fp.mrf.200, fs.freq.fp.mrf.250, fs.freq.fp.mrf.300)

  print("Self-cons test..")
  for(i in 1:length(sc.st)) {
    fracs <- lapply(sc.st[[i]], FUN = function(x) length(x)/nrow(filter_data))
    p <- mean(as.numeric(fracs))
    print(paste(" - p:",p,sep=""))
    print(length(sc.st.freq[[i]]))
    sc.st.freq[[i]] <- na.omit(sc.st.freq[[i]])
    print(length(sc.st.freq[[i]]))
    for(j in 1:(length(sc.st.freq[[i]]))) {
      if(sc.st.freq[[i]][j] > 0) {
        bt <- binom.test(x = sc.st.freq[[i]][j], n = nboot, p = p)
        stability.feats[j,i] <- bt$p.value
      }
      else
        stability.feats[j,i] <- 1
    }
  }
  acc.df <- data.frame(acc.fp.mrf.all, acc.fp.mrf.30,
                       acc.fp.mrf.50, acc.fp.mrf.100,
                       acc.fp.mrf.150, acc.fp.mrf.200,
                       acc.fp.mrf.250, acc.fp.mrf.300)
  gm.df  <- data.frame(gmean.fp.mrf.all, gmean.fp.mrf.30,
                       gmean.fp.mrf.50, gmean.fp.mrf.100,
                       gmean.fp.mrf.150, gmean.fp.mrf.200,
                       gmean.fp.mrf.250, gmean.fp.mrf.300)
  fsc.df <- data.frame(fscore.fp.mrf.all, fscore.fp.mrf.30,
                       fscore.fp.mrf.50, fscore.fp.mrf.100,
                       fscore.fp.mrf.150, fscore.fp.mrf.200,
                       fscore.fp.mrf.250, fscore.fp.mrf.300)
  sel.feats <- data.frame(size.fs.fp)
  return(list(stability.feats, list.fs.fp.mrf.all,
              st.df, sc.st, sc.st.freq, acc.df,
              gm.df, fsc.df, sel.feats))
}

## Preparación de los datos para aplicar el algoritmo

class_fprf <- data.frame(label.train)
colnames(class_fprf)[1] <- "class"
rownames(class_fprf) <- rownames(data.train)
```

```
data_FPRF <- ExpressionSet(
  as.matrix(t(data.train)),
  phenoData = AnnotatedDataFrame(class_fprf))

## Aplicación del algoritmo

set.seed(seed.classifier)
FPRF_results <- boot.sam.eval(filter_data=data_FPRF,nboot=5)

##FPRF,tabla y extracción de genes
accuracy_FPRF <- data.frame(t(unlist(FPRF_results[[6]])))
accuracy_FPRF_tab <- list()
accuracy_FPRF_tab$"All" <- sum(accuracy_FPRF[1:5])/5
accuracy_FPRF_tab$"30" <- sum(accuracy_FPRF[6:10])/5
accuracy_FPRF_tab$"50" <- sum(accuracy_FPRF[11:15])/5
accuracy_FPRF_tab$"100" <- sum(accuracy_FPRF[16:20])/5
accuracy_FPRF_tab$"150" <- sum(accuracy_FPRF[21:25])/5
accuracy_FPRF_tab$"200" <- sum(accuracy_FPRF[26:30])/5
accuracy_FPRF_tab$"250" <- sum(accuracy_FPRF[31:35])/5
accuracy_FPRF_tab$"300" <- sum(accuracy_FPRF[36:40])/5

accuracy_sd_FPRF_tab <- list()

accuracy_sd_FPRF_tab$"All" <- sd(accuracy_FPRF[1:5])
accuracy_sd_FPRF_tab$"30" <- sd(accuracy_FPRF[6:10])
accuracy_sd_FPRF_tab$"50" <- sd(accuracy_FPRF[11:15])
accuracy_sd_FPRF_tab$"100" <- sd(accuracy_FPRF[16:20])
accuracy_sd_FPRF_tab$"150" <- sd(accuracy_FPRF[21:25])
accuracy_sd_FPRF_tab$"200" <- sd(accuracy_FPRF[26:30])
accuracy_sd_FPRF_tab$"250" <- sd(accuracy_FPRF[31:35])
accuracy_sd_FPRF_tab$"300" <- sd(accuracy_FPRF[36:40])

metrics_FPRF <-rbind(data.frame(accuracy_FPRF_tab),data.frame(accuracy_sd_FPRF_tab))
colnames(metrics_FPRF) <- c("All","30","50","100","150","200","250","300")
rownames(metrics_FPRF) <- c("Accuracy", "Accuracy SD")
metrics_FPRF <- round(metrics_FPRF,3)
genes_FPRF <- unlist(FPRF_results[[4]][2][[1]][1])
genes_FPRF <- gsub("X",replacement="",x=genes_FPRF)

##SVM rfe

subset <- c(3000,2000,1500,1000,750,500,250,200,150,100,80,60,40,30,20,10,5)
FSctrl <- rfeControl(functions = caretFuncs,
                     method = "cv",
                     number =5,
                     rerank = TRUE,
                     saveDetails = FALSE,
                     verbose = TRUE)
TRctrl = trainControl(method = "cv",
                      number = 10)

set.seed(seed.classifier)
```

```
svmRFE_model <- rfe(data.train,label.train,
                    sizes = subset,
                    rfeControl = FSctrl,
                    method = "svmLinear",
                    trControl = TRctrl)

##svmRFE_model,tabla y extracción de genes
svmRFE_model$bestSubset
genes_svmrfe <- svmRFE_model$optVariables

accuracy_svmRFE <- svmRFE_model$results$Accuracy
accuracy_sd_svmRFE <- svmRFE_model$results$AccuracySD
metric_svmRFE <- rbind(accuracy_svmRFE,accuracy_sd_svmRFE)
colnames(metric_svmRFE) <-
  rev(c("3652","3000","2000",
        "1500","1000","750",
        "500","250","200","150",
        "100","80","60","40",
        "30","20","10","5"))
rownames(metric_svmRFE) <- c("Accuracy", "Accuracy SD")
metric_svmRFE <- round(metric_svmRFE,3)

##rfRFE

subset <- c(3000,2000,1500,1000,750,500,250,200,150,100,80,60,40,30,20,10,5)
mtry <- sqrt(ncol(data.train))
FSctrl <- rfeControl(functions = rfFuncs,
                     method = "cv",
                     number =5,
                     rerank = TRUE,
                     saveDetails = FALSE,
                     verbose = TRUE,
                     returnResamp="all")
TRctrl = trainControl(method = "cv",
                      number = 10)
tunegrid <- expand.grid(.mtry=mtry)

set.seed(seed.classifier)
rfRFE_model <- rfe(data.train,label.train,
                   sizes = subset,
                   rfeControl = FSctrl,
                   ## Options to train()
                   method = "rf",
                   ## Inner resampling process
                   trControl = TRctrl,
                   tuneGrid = tunegrid,
                   ntree=1000)

##rfRFE_model,tabla y extracción de genes
rfRFE_model$bestSubset
genes_rfrfe <- rfRFE_model$optVariables
```

```
accuracy_rfRFE <- rfRFE_model$results$Accuracy
accuracy_sd_rfRFE <- rfRFE_model$results$AccuracySD
metric_rfRFE <- rbind(accuracy_rfRFE,accuracy_sd_rfRFE)
colnames(metric_rfRFE) <- rev(c("3652","3000","2000",
                                "1500","1000","750","500",
                                "250","200","150",
                                "100","80","60","40",
                                "30","20","10","5"))
rownames(metric_rfRFE) <- c("Accuracy", "Accuracy SD")
metric_rfRFE <- round(metric_rfRFE,3)
metric_rfRFE

##Reducción de la dimensión con autoencoders
library(ggplot2)
library(keras) # high level wrapping interface
K <- keras::backend() # The choosen backend "TensorFlow"
library(tidyverse)

###Normalización de los datos
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}
data_nrm <- (apply(correlation_filter,2, normalize))
data_nrm <- as.data.frame(data_nrm)
data_nrm <-round(data_nrm,3)

##Seteamos train y data test para el autoencoder
data.ae <- data_nrm[train,]
set.seed(seed.partition)
n_train_ae <- 0.7
n_ae <- nrow(data.ae)
train.ae <- sample(n_ae,floor(n_ae*n_train_ae))
data.train.ae <- data.ae[train.ae,]
data.test.ae <- data.ae[-train.ae,]
data.train.ae <- data.train.ae %>% as.matrix()
data.test.ae <- data.test.ae %>% as.matrix()


###Configuración de la red neuronal del autoencoder (y decoder)
k_clear_session()
K <- keras::backend() # The choosen backend "TensorFlow"
dim.data <- ncol(data_nrm)
input_layer <-
  layer_input(shape = dim.data)


inter2 <- layer_dense(input_layer, units = 400, activation = "relu") %>%
  layer_dropout(rate = 0.1)

encoder <- layer_dense(inter2, units = 50,activation ="relu")

expan2 <- layer_dense(encoder, units = 400, activation = "relu") %>%
```

```
  layer_dropout(rate = 0.1)

decoder <- layer_dense(expan2, units = dim.data, activation = "sigmoid")


autoencoder_model <- keras_model(inputs = input_layer, outputs = decoder)

autoencoder_model %>% compile(
  loss='mean_squared_error',
  optimizer='adam',
  metrics = c('accuracy')
)

summary(autoencoder_model)

##Autoencoder sobre los datos

history <-
  autoencoder_model %>%
  keras::fit(data.train.ae,
             data.train.ae,
             epochs=500,
             shuffle=TRUE,
             validation_data= list(data.test.ae, data.test.ae)
  )

##Graficamos loss y accuracy del modelo
plot(history,metrics="loss")+
  theme_bw()+
  theme(axis.title.x=element_text(size=14),
        axis.title.y=element_text(size=14),
        axis.text.y = element_text(size=10),
        axis.text.x = element_text(size=10),
        legend.position = c(0.15,0.85),
        legend.title = element_blank(),
        legend.text = element_text(size=14),
        strip.background = element_blank(),
        strip.text.y = element_blank())+
  ylim(0.00,0.015)+
  ylab("Error cuadrático medio")+
  scale_fill_manual(values=c("#CC916F", "#0F2266"))+
  scale_color_manual(values=c("#CC916F", "#0F2266"))

plot(history,metrics="acc")+
  theme_bw()+
  theme(axis.title.x=element_text(size=14),
        axis.title.y=element_text(size=14),
        axis.text.y = element_text(size=10),
        axis.text.x = element_text(size=10),
        legend.position = c(0.15,0.85),
        legend.title = element_blank(),
        legend.text = element_text(size=14),
```

```
        strip.background = element_blank(),
        strip.text.y = element_blank())+
  ylim(0.05,0.17)+
  ylab("Precisión")+
  scale_fill_manual(values=c("#CC916F", "#0F2266"))+
  scale_color_manual(values=c("#CC916F", "#0F2266"))


##Reconstruimos los datos originales con el modelo aprendido
reconstructed_points <-
  autoencoder_model %>%
  keras::predict_on_batch(x = as.matrix(data_nrm))

##Preparación de los datos para su visualización
##Construye un data set con los puntos originales
#y el valor de los puntos reconstruidosen el autoencoder
Viz_data <-
  dplyr::bind_rows(
    reconstructed_points %>%
      tibble::as_tibble() %>%
      setNames(names(as.matrix(data_nrm) %>% tibble::as_tibble())) %>%
      dplyr::mutate(data_origin = "reconstruídos"),
    as.matrix(data_nrm) %>%
      tibble::as_tibble() %>%
      dplyr::mutate(data_origin = "originales")
  )

#Visualizamos varios pares de datos para ver los originales vs reconstruidos
Viz_data %>%
  ggplot(aes(`245304_at`,`245281_at`, color = data_origin))+
  geom_point()+
  scale_color_manual(values=c("#96D489", "#7F5339"))+
  theme_bw()+
  theme(axis.title.x=element_text(size=14),
        axis.title.y=element_text(size=14),
        axis.text.y = element_text(size=10),
        axis.text.x = element_text(size=10),
        legend.position = c(0.15,0.85),
        legend.title = element_blank(),
        legend.text = element_text(size=14))+
  ylab("245281_at")+
  xlab("245304_at")

Viz_data %>%
  ggplot(aes(`244972_at`,`245021_at`, color = data_origin))+
  geom_point()+
  scale_color_manual(values=c("#96D489", "#7F5339"))+
  theme_bw()+
  theme(axis.title.x=element_text(size=14),
        axis.title.y=element_text(size=14),
        axis.text.y = element_text(size=10),
        axis.text.x = element_text(size=10),
```

```
        legend.position = c(0.15,0.85),
        legend.title = element_blank(),
        legend.text = element_text(size=14))+
  xlab("244972_at")+
  ylab("245021_at")


##Guardamos los pesos del autoencoder
autoencoder_weights <-
  autoencoder_model %>%
  keras::get_weights()

keras::save_model_weights_hdf5(object = autoencoder_model,
                               filepath = 'weights.hdf5',
                               overwrite = TRUE)

##Aplicar sólo encoder sobre todos los datos
encoder_model <- keras_model(inputs = input_layer, outputs = encoder)

encoder_model %>% compile(
  loss='mean_squared_error',
  optimizer='adam',
  metrics = c('accuracy')
)


embeded_points <-
  encoder_model %>%
  keras::predict_on_batch(x = as.matrix(data_nrm))

##Obtener los datos de dimensión reducida
reduce_data.ae<- data.frame(embeded_points)
rownames(reduce_data.ae) <- rownames(data_nrm)

###Extracción de genes seleccionados en datos originales

###Para algoritmos rfe y svm
data_genes_svm <- correlation_filter[,genes_svmrfe]
data_genes_rf <- correlation_filter[,genes_rfrfe]
data_genes_fprf <- correlation_filter[,genes_FPRF]

###Para red neuronal (normalizados)

data_norm_genes_svm <- data_nrm[,genes_svmrfe]
data_norm_genes_rf <- data_nrm[,genes_rfrfe]
data_norm_genes_fprf <- data_nrm[,genes_FPRF]

###Para todos
dim(reduce_data.ae)

###Algoritmos de machine learning para clasificación
##Train y test data sets
```

```
data_genes_svm_train <- data_genes_svm[train,]
data_genes_svm_test <- data_genes_svm[-train,]
data_genes_rf_train <- data_genes_rf[train,]
data_genes_rf_test <- data_genes_rf[-train,]
data_genes_fprf_train <- data_genes_fprf[train,]
data_genes_fprf_test <- data_genes_fprf[-train,]
data_norm_train <- data_nrm[train,]
data_norm_test <- data_nrm[-train,]
data_norm_genes_svm_train <- data_norm_genes_svm[train,]
data_norm_genes_svm_test <- data_norm_genes_svm[-train,]
data_norm_genes_rf_train <- data_norm_genes_rf[train,]
data_norm_genes_rf_test <- data_norm_genes_rf[-train,]
data_norm_genes_fprf_train <- data_norm_genes_fprf[train,]
data_norm_genes_fprf_test <- data_norm_genes_fprf[-train,]
reduce_data.ae_train <- reduce_data.ae[train,]
reduce_data.ae_test <- reduce_data.ae[-train,]

label.test[104] <- "root"

##Función para evaluar medidas de manera personalizada
#a lo largo de las iteraciones de caret
##De esta manera devuelve la media de estas medidas en las X-fold CV.
##F-score,G-mean,Accuracy
new_metrics <- function(data, lev = NULL, model = NULL) {
  cm <- table(data$pred,data$obs)
  ## Number of groups
  Ngp <- nrow(cm)

  ## Total : TP + TN + FP + FN
  Tot <- sum(cm)

  ## TP : True positive item : All items on diagonal
  TP <- diag(cm)

  ## TP + TN : sum of diagonal = All correct identification
  TP_TN <- sum(TP)

  ## TP + FP : sum of columns : Automatic classification
  TP_FP <- colSums(cm)

  ## TP + FN : sum of rows : Manual classification
  TP_FN <- rowSums(cm)

  ## FP : False positive items
  FP <- TP_FP - TP

  ## FN : False negative item
  FN <- TP_FN - TP

  ## TN : True Negative = Total - TP - FP - FN
  TN <- rep(Tot, Ngp) - TP - FP - FN
```

```
  ## The 8 basic ratios
  ## Recall = TP / (TP + FN) = 1 - FNR
  Recall <- TP / (TP_FN)

  ## Specificity = TN / (TN + FP) = 1 - FPR
  Specificity <- TN / (TN + FP)

  ## Precision = TP / (TP + FP) = 1 - FDR
  Precision <- TP / (TP_FP)

  ## F-score = F-measure = F1 score = Harmonic mean of Precision and recall
  Fmeasure <- 2 * ((Precision * Recall) / (Precision + Recall))
  Fmeasure[is.nan(Fmeasure)] <- 0
  Fscore <- mean(Fmeasure)
  Gmean <- prod(Recall)^(1/Ngp)
  Acc1 <- ((TP + TN) / (TP + TN + FP + FN))
  Acc <- mean(Acc1)
  c(Accuracy=Acc,FScore = Fscore,GMean = Gmean)
}
##Clasificación por SVM

#Con todos los genes
TRctrl = trainControl(method = "cv",
                      number = 5,
                      summaryFunction = new_metrics,
                      classProbs = TRUE)
set.seed(seed.classifier)
svm_all_train <- caret::train(data.train,label.train,
                              method='svmLinear',tuneGrid= NULL,
                              trace = FALSE, trControl = TRctrl)
set.seed(seed.classifier)
svm_all_test <- caret::train(data.test,label.test,
                             method='svmLinear',tuneGrid= NULL,
                             trace = FALSE, trControl = TRctrl)

#Genes por svmRFE
set.seed(seed.classifier)
svm_svmrfe_train <- caret::train(data_genes_svm_train,
                                 label.train, method='svmLinear',
                                 tuneGrid= NULL,
                                 trace = FALSE, trControl = TRctrl)
set.seed(seed.classifier)
svm_svmrfe_test <- caret::train(data_genes_svm_test,
                                label.test, method='svmLinear',
                                tuneGrid= NULL, trace = FALSE,
                                trControl = TRctrl)

#Genes por rfRFE
set.seed(seed.classifier)
svm_rfrfe_train <- caret::train(data_genes_rf_train, label.train, method='svmLinear',
                            tuneGrid= NULL, trace = FALSE, trControl = TRctrl)
set.seed(seed.classifier)
```

```
svm_rfrfe_test <- caret::train(data_genes_rf_test, label.test,
                               method='svmLinear',
                               tuneGrid= NULL, trace = FALSE,
                               trControl = TRctrl)

#Genes por FPRF
set.seed(seed.classifier)
svm_fprf_train <- caret::train(data_genes_fprf_train,
                               label.train, method='svmLinear',
                               tuneGrid= NULL,
                               trace = FALSE, trControl = TRctrl)
set.seed(seed.classifier)
svm_fprf_test <- caret::train(data_genes_fprf_test,
                              label.test, method='svmLinear',
                              tuneGrid= NULL,
                              trace = FALSE, trControl = TRctrl)

#Componentes por autoencoder
set.seed(seed.classifier)
svm_autoencoder_train <- caret::train(
  reduce_data.ae_train, label.train, method='svmLinear',
                        tuneGrid= NULL,
                        trace = FALSE,
                        trControl = TRctrl)
set.seed(seed.classifier)
svm_autoencoder_test <- caret::train(reduce_data.ae_test,
                                     label.test, method='svmLinear',
                                     tuneGrid= NULL, trace = FALSE,
                                     trControl = TRctrl)

##Clasificación por randomforest
#Con todos los genes
set.seed(seed.classifier)
rf_all_train <- caret::train(data.train,label.train, method = "rf",
                      trControl = TRctrl,
                      ntree=1000,
                      tuneLength=1)
set.seed(seed.classifier)
rf_all_test <- caret::train(data.test,label.test, method = "rf",
                            trControl = TRctrl,
                            ntree=1000,
                           tuneLength=1)
#Genes por svmRFE
set.seed(seed.classifier)
rf_svmrfe_train <- caret::train(data_genes_svm_train,label.train, method = "rf",
                         trControl = TRctrl,
                         ntree=1000,
                         tuneLength=1)
set.seed(seed.classifier)
rf_svmrfe_test <- caret::train(data_genes_svm_test,label.test, method = "rf",
                         trControl = TRctrl,
                         ntree=1000,
```

```
                          tuneLength=1)

#Genes por rfRFE
set.seed(seed.classifier)
rf_rfrfe_train <- caret::train(data_genes_rf_train,label.train, method = "rf",
                        trControl = TRctrl,
                        ntree=1000,
                        tuneLength=1)
set.seed(seed.classifier)
rf_rfrfe_test <- caret::train(data_genes_rf_test,label.test, method = "rf",
                               trControl = TRctrl,
                               ntree=1000,
                              tuneLength=1)

#Genes por FPRF
set.seed(seed.classifier)
rf_fprf_train <- caret::train(data_genes_fprf_train,label.train, method = "rf",
                        trControl = TRctrl,
                        ntree=1000,
                       tuneLength=1)
set.seed(seed.classifier)
rf_fprf_test <- caret::train(data_genes_fprf_test,label.test, method = "rf",
                              trControl = TRctrl,
                              ntree=1000,
                              tuneLength=1)

#Componentes por autoencoder
set.seed(seed.classifier)
rf_autoencoder_train <- caret::train(reduce_data.ae_train,label.train, method = "rf",
                       trControl = TRctrl,
                       ntree=1000,
                       tuneLength=1)
set.seed(seed.classifier)
rf_autoencoder_test <- caret::train(reduce_data.ae_test,label.test, method = "rf",
                               trControl = TRctrl,
                               ntree=1000,
                               tuneLength=1)

#Clasificación por redes neuronales, con datasets normalizados
TRctrl = trainControl(method = "cv",
                      number = 3,
                      summaryFunction = new_metrics,
                      classProbs = TRUE)

#Con todos los genes
t.grid_1=expand.grid(size=1,decay=0.1)
set.seed(seed.classifier)
nn_all_train <- caret::train(data_norm_train,label.train, method = "nnet",
                             trControl = TRctrl,
                             tuneGrid=t.grid_1,
                             MaxNWts = 3663)
set.seed(seed.classifier)
```

```
nn_all_test <- caret::train(data_norm_test,label.test, method = "nnet",
                            trControl = TRctrl,
                           tuneGrid=t.grid_1,
                           MaxNWts = 3663)

#Genes FPRF
t.grid=expand.grid(size=3,decay=0.1)
set.seed(seed.classifier)
nn_fprf_train <- caret::train(data_norm_genes_fprf_train,label.train, method = "nnet",
                            trControl = TRctrl,tuneGrid=t.grid)
set.seed(seed.classifier)
nn_fprf_test <- caret::train(data_norm_genes_fprf_test,label.test, method = "nnet",
                            trControl = TRctrl,tuneGrid=t.grid)

#Genes SVMRFE
set.seed(seed.classifier)
nn_svmrfe_train <- caret::train(data_norm_genes_svm_train,label.train, method = "nnet",
                             trControl = TRctrl,tuneGrid=t.grid)
set.seed(seed.classifier)
nn_svmrfe_test <- caret::train(data_norm_genes_svm_test,label.test, method = "nnet",
                            trControl = TRctrl,tuneGrid=t.grid)

#Genes RFRFE
set.seed(seed.classifier)
nn_rfrfe_train <- caret::train(data_norm_genes_rf_train,label.train, method = "nnet",
                               trControl = TRctrl,tuneGrid=t.grid)
set.seed(seed.classifier)
nn_rfrfe_test <- caret::train(data_norm_genes_rf_test,label.test, method = "nnet",
                              trControl = TRctrl,tuneGrid=t.grid)
#Dimensiones encoder
set.seed(seed.classifier)
nn_ae_train <- caret::train(reduce_data.ae_train,label.train, method = "nnet",
                              trControl = TRctrl,tuneGrid=t.grid)
set.seed(seed.classifier)
nn_ae_test <- caret::train(reduce_data.ae_test,label.test, method = "nnet",
                             trControl = TRctrl,tuneGrid=t.grid)
#Clasificación por redes neuronales, con datasets normalizados
TRctrl = trainControl(method = "cv",
                      number = 3,
                      summaryFunction = new_metrics,
                      classProbs = TRUE)

#Con todos los genes
t.grid_1=expand.grid(size=1,decay=0.1)
set.seed(seed.classifier)
nn_all_train <- caret::train(data_norm_train,label.train, method = "nnet",
                            trControl = TRctrl,
                            tuneGrid=t.grid_1,
                            MaxNWts = 3663)
set.seed(seed.classifier)
nn_all_test <- caret::train(data_norm_test,label.test, method = "nnet",
                            trControl = TRctrl,
```

```
                              tuneGrid=t.grid_1,
                              MaxNWts = 3663)

#Genes FPRF
t.grid=expand.grid(size=3,decay=0.1)
set.seed(seed.classifier)
nn_fprf_train <- caret::train(data_norm_genes_fprf_train,label.train, method = "nnet",
                              trControl = TRctrl,tuneGrid=t.grid)
set.seed(seed.classifier)
nn_fprf_test <- caret::train(data_norm_genes_fprf_test,label.test, method = "nnet",
                              trControl = TRctrl,tuneGrid=t.grid)

#Genes SVMRFE
set.seed(seed.classifier)
nn_svmrfe_train <- caret::train(data_norm_genes_svm_train,label.train, method = "nnet",
                               trControl = TRctrl,tuneGrid=t.grid)
set.seed(seed.classifier)
nn_svmrfe_test <- caret::train(data_norm_genes_svm_test,label.test, method = "nnet",
                              trControl = TRctrl,tuneGrid=t.grid)

#Genes RFRFE
set.seed(seed.classifier)
nn_rfrfe_train <- caret::train(data_norm_genes_rf_train,label.train, method = "nnet",
                                trControl = TRctrl,tuneGrid=t.grid)
set.seed(seed.classifier)
nn_rfrfe_test <- caret::train(data_norm_genes_rf_test,label.test, method = "nnet",
                               trControl = TRctrl,tuneGrid=t.grid)
#Dimensiones encoder
set.seed(seed.classifier)
nn_ae_train <- caret::train(reduce_data.ae_train,label.train, method = "nnet",
                               trControl = TRctrl,tuneGrid=t.grid)
set.seed(seed.classifier)
nn_ae_test <- caret::train(reduce_data.ae_test,label.test, method = "nnet",
                               trControl = TRctrl,tuneGrid=t.grid)

###Resumen medidas
mc <- list()
mc$Accuracy <- c(svm_all_train$results$Accuracy,svm_all_test$results$Accuracy,
               svm_fprf_train$results$Accuracy,svm_fprf_test$results$Accuracy,
               svm_svmrfe_train$results$Accuracy,svm_svmrfe_test$results$Accuracy,
               svm_rfrfe_train$results$Accuracy,svm_rfrfe_test$results$Accuracy,
               svm_autoencoder_train$results$Accuracy,
               svm_autoencoder_test$results$Accuracy,
               rf_all_train$results$Accuracy,rf_all_test$results$Accuracy,
               rf_fprf_train$results$Accuracy,rf_fprf_test$results$Accuracy,
               rf_svmrfe_train$results$Accuracy,rf_svmrfe_test$results$Accuracy,
               rf_rfrfe_train$results$Accuracy,rf_rfrfe_test$results$Accuracy,
               rf_autoencoder_train$results$Accuracy,
               rf_autoencoder_test$results$Accuracy,
               nn_all_train$results$Accuracy,nn_all_test$results$Accuracy,
               nn_fprf_train$results$Accuracy,nn_fprf_test$results$Accuracy,
               nn_svmrfe_train$results$Accuracy,nn_svmrfe_test$results$Accuracy,
```

```
                nn_rfrfe_train$results$Accuracy,nn_rfrfe_test$results$Accuracy,
                nn_ae_train$results$Accuracy,nn_ae_test$results$Accuracy)
mc$AccSD <- c(svm_all_train$results$AccuracySD,svm_all_test$results$AccuracySD,
                 svm_fprf_train$results$AccuracySD,
              svm_fprf_test$results$AccuracySD,
                 svm_svmrfe_train$results$AccuracySD,
              svm_svmrfe_test$results$AccuracySD,
                 svm_rfrfe_train$results$AccuracySD,
              svm_rfrfe_test$results$AccuracySD,
                 svm_autoencoder_train$results$AccuracySD,
              svm_autoencoder_test$results$AccuracySD,
                 rf_all_train$results$AccuracySD,
              rf_all_test$results$AccuracySD,
                 rf_fprf_train$results$AccuracySD,
              rf_fprf_test$results$AccuracySD,
                 rf_svmrfe_train$results$AccuracySD,
              rf_svmrfe_test$results$AccuracySD,
                 rf_rfrfe_train$results$AccuracySD,
              rf_rfrfe_test$results$AccuracySD,
                 rf_autoencoder_train$results$AccuracySD,
              rf_autoencoder_test$results$AccuracySD,
                 nn_all_train$results$AccuracySD,nn_all_test$results$AccuracySD,
                 nn_fprf_train$results$AccuracySD,nn_fprf_test$results$AccuracySD,
                 nn_svmrfe_train$results$AccuracySD,nn_svmrfe_test$results$AccuracySD,
                 nn_rfrfe_train$results$AccuracySD,nn_rfrfe_test$results$AccuracySD,
                 nn_ae_train$results$AccuracySD,nn_ae_test$results$AccuracySD)

mc$Fscore <- c(svm_all_train$results$FScore,svm_all_test$results$FScore,
                svm_fprf_train$results$FScore,svm_fprf_test$results$FScore,
                svm_svmrfe_train$results$FScore,svm_svmrfe_test$results$FScore,
                svm_rfrfe_train$results$FScore,svm_rfrfe_test$results$FScore,
                svm_autoencoder_train$results$FScore,svm_autoencoder_test$results$FScore,
                rf_all_train$results$FScore,rf_all_test$results$FScore,
                rf_fprf_train$results$FScore,rf_fprf_test$results$FScore,
                rf_svmrfe_train$results$FScore,rf_svmrfe_test$results$FScore,
                rf_rfrfe_train$results$FScore,rf_rfrfe_test$results$FScore,
                rf_autoencoder_train$results$FScore,rf_autoencoder_test$results$FScore,
                nn_all_train$results$FScore,nn_all_test$results$FScore,
                nn_fprf_train$results$FScore,nn_fprf_test$results$FScore,
                nn_svmrfe_train$results$FScore,nn_svmrfe_test$results$FScore,
                nn_rfrfe_train$results$FScore,nn_rfrfe_test$results$FScore,
                nn_ae_train$results$FScore,nn_ae_test$results$FScore)
mc$Fscore <- c(svm_all_train$results$FScore,svm_all_test$results$FScore,
                svm_fprf_train$results$FScore,svm_fprf_test$results$FScore,
                svm_svmrfe_train$results$FScore,svm_svmrfe_test$results$FScore,
                svm_rfrfe_train$results$FScore,svm_rfrfe_test$results$FScore,
                svm_autoencoder_train$results$FScore,svm_autoencoder_test$results$FScore,
                rf_all_train$results$FScore,rf_all_test$results$FScore,
                rf_fprf_train$results$FScore,rf_fprf_test$results$FScore,
                rf_svmrfe_train$results$FScore,rf_svmrfe_test$results$FScore,
                rf_rfrfe_train$results$FScore,rf_rfrfe_test$results$FScore,
                rf_autoencoder_train$results$FScore,rf_autoencoder_test$results$FScore,
```

```
                nn_all_train$results$FScore,nn_all_test$results$FScore,
                nn_fprf_train$results$FScore,nn_fprf_test$results$FScore,
                nn_svmrfe_train$results$FScore,nn_svmrfe_test$results$FScore,
                nn_rfrfe_train$results$FScore,nn_rfrfe_test$results$FScore,
                nn_ae_train$results$FScore,nn_ae_test$results$FScore)
mc$FSD <- c(svm_all_train$results$FScoreSD,svm_all_test$results$FScoreSD,
                svm_fprf_train$results$FScoreSD,svm_fprf_test$results$FScoreSD,
                svm_svmrfe_train$results$FScoreSD,svm_svmrfe_test$results$FScoreSD,
                svm_rfrfe_train$results$FScoreSD,svm_rfrfe_test$results$FScoreSD,
                svm_autoencoder_train$results$FScoreSD,svm_autoencoder_test$results$FScoreSD,
                rf_all_train$results$FScoreSD,rf_all_test$results$FScoreSD,
                rf_fprf_train$results$FScoreSD,rf_fprf_test$results$FScoreSD,
                rf_svmrfe_train$results$FScoreSD,rf_svmrfe_test$results$FScoreSD,
                rf_rfrfe_train$results$FScoreSD,rf_rfrfe_test$results$FScoreSD,
                rf_autoencoder_train$results$FScoreSD,rf_autoencoder_test$results$FScoreSD,
                nn_all_train$results$FScoreSD,nn_all_test$results$FScoreSD,
                nn_fprf_train$results$FScoreSD,nn_fprf_test$results$FScoreSD,
                nn_svmrfe_train$results$FScoreSD,nn_svmrfe_test$results$FScoreSD,
                nn_rfrfe_train$results$FScoreSD,nn_rfrfe_test$results$FScoreSD,
                nn_ae_train$results$FScoreSD,nn_ae_test$results$FScoreSD)
mc$GMean <- c(svm_all_train$results$GMean,svm_all_test$results$GMean,
             svm_fprf_train$results$GMean,svm_fprf_test$results$GMean,
             svm_svmrfe_train$results$GMean,svm_svmrfe_test$results$GMean,
             svm_rfrfe_train$results$GMean,svm_rfrfe_test$results$GMean,
             svm_autoencoder_train$results$GMean,svm_autoencoder_test$results$GMean,
             rf_all_train$results$GMean,rf_all_test$results$GMean,
             rf_fprf_train$results$GMean,rf_fprf_test$results$GMean,
             rf_svmrfe_train$results$GMean,rf_svmrfe_test$results$GMean,
             rf_rfrfe_train$results$GMean,rf_rfrfe_test$results$GMean,
             rf_autoencoder_train$results$GMean,rf_autoencoder_test$results$GMean,
             nn_all_train$results$GMean,nn_all_test$results$GMean,
             nn_fprf_train$results$GMean,nn_fprf_test$results$GMean,
             nn_svmrfe_train$results$GMean,nn_svmrfe_test$results$GMean,
             nn_rfrfe_train$results$GMean,nn_rfrfe_test$results$GMean,
             nn_ae_train$results$GMean,nn_ae_test$results$GMean)
mc$GSD <- c(svm_all_train$results$GMeanSD,svm_all_test$results$GMeanSD,
               svm_fprf_train$results$GMeanSD,svm_fprf_test$results$GMeanSD,
               svm_svmrfe_train$results$GMeanSD,svm_svmrfe_test$results$GMeanSD,
               svm_rfrfe_train$results$GMeanSD,svm_rfrfe_test$results$GMeanSD,
               svm_autoencoder_train$results$GMeanSD,svm_autoencoder_test$results$GMeanSD,
               rf_all_train$results$GMeanSD,rf_all_test$results$GMeanSD,
               rf_fprf_train$results$GMeanSD,rf_fprf_test$results$GMeanSD,
               rf_svmrfe_train$results$GMeanSD,rf_svmrfe_test$results$GMeanSD,
               rf_rfrfe_train$results$GMeanSD,rf_rfrfe_test$results$GMeanSD,
               rf_autoencoder_train$results$GMeanSD,rf_autoencoder_test$results$GMeanSD,
               nn_all_train$results$GMeanSD,nn_all_test$results$GMeanSD,
               nn_fprf_train$results$GMeanSD,nn_fprf_test$results$GMeanSD,
               nn_svmrfe_train$results$GMeanSD,nn_svmrfe_test$results$GMeanSD,
               nn_rfrfe_train$results$GMeanSD,nn_rfrfe_test$results$GMeanSD,
               nn_ae_train$results$GMeanSD,nn_ae_test$results$GMeanSD)

mc<-data.frame(mc)
```

```
mc$Group <- rep(c("train","test"),15)
mc$MSel <- rep(c("Todas", "Todas", "FP-RF","FP-RF","SVM-RFE","SVM-RFE",
                 "RF-RFE","RF-RFE","Encoder","Encoder"),3)
mc$Mclas <- c(rep("SVM",10),rep("RF",10),rep("ANN",10))

mc[,1:6] <- round(mc[,1:6],2)
mc$MSel <- factor(mc$MSel , levels = c("Todas","FP-RF", "SVM-RFE", "RF-RFE","Encoder"))
mc<-mc[order(mc$MSel),]

save(mc,file=file.path(results,"mc.RData"))


#Análisis funcional de los genes

##Anotación

anotacion <- read.csv(file.path(rawDir,"allprobe.txt"),sep="\t",header=TRUE)
###Importar archivo con anotación

colnames(anotacion) <- c("Sonda","AGI","Símbolo","Descripción")

#Anotación
anot_fprf <- data.frame('Sonda'=genes_FPRF)
anot_rfrfe <- data.frame("Sonda"=genes_rfrfe)
anot_svmrfe <- data.frame("Sonda"=genes_svmrfe)

anot_fprf <- plyr::join(anotacion,anot_fprf,by='Sonda',type='right')
anot_rfrfe <- plyr::join(anotacion,anot_rfrfe,by='Sonda',type='right')
anot_svmrfe <- plyr::join(anotacion,anot_svmrfe,by='Sonda',type='right')


##Heatmap de las variables seleccionadas

###Lectura de datos

data_cluster_1<- data.frame(t(data_genes_fprf))
data_cluster_2<- data.frame(t(data_genes_rf))
data_cluster_3<- data.frame(t(data_genes_svm))

###Anotación clases
grupo_muestras <- data.frame(class)
colnames(grupo_muestras) <- "Clase"
rownames(grupo_muestras) <- colnames(data_cluster_centrada)
grupo_muestras["seedling_48.CEL",] <- "root"

colores_grupo <- list(Clase=c("seedling" = "#F03B0E", "root" = "#36D6DA",
                              "seed" = "#F7B409","flower"="#8B1C42",
                              "leaf"= "#0A4A75"))

###Funcion escalado
cal_z_score <- function(x){
  (x - mean(x)) / sd(x)
```

```
}
data_cluster1_centrada <- t(apply(data_cluster_1, 1, cal_z_score))
data_cluster2_centrada <- t(apply(data_cluster_2, 1, cal_z_score))
data_cluster3_centrada <- t(apply(data_cluster_3, 1, cal_z_score))

###Calculo distancia y cluster
hr1 <- hclust(dist(data_cluster1_centrada,method = "euclidean"),
              method="complete") ##Cluster 1 3 grupos
hr2 <- hclust(dist(data_cluster2_centrada,method = "euclidean"),
              method="complete") ##Cluster fila 5
hr3 <- hclust(dist(data_cluster3_centrada,method = "euclidean"),
              method="complete") ##Cluster fila 6

###Heatmaps con pheatmap

pheatmap(data_cluster1_centrada,
         color=colorRampPalette((brewer.pal(n = 11, name = "PuOr")))(100),
         annotation_colors = colores_grupo,
         annotation_col = grupo_muestras,#anotacion fila
         cutree_rows = 3, #Cortes por grupo
         show_rownames = FALSE,
         show_colnames = FALSE,
         ## clustering_distance_rows = "euclidean",
         ## clustering_method = "complete",
         annotation_names_col=F
)

pheatmap(data_cluster2_centrada,
         color=colorRampPalette((brewer.pal(n = 11, name = "PuOr")))(100),
         annotation_colors = colores_grupo,
         annotation_col = grupo_muestras,#anotacion fila
         cutree_rows = 5, #Cortes por grupo
         show_rownames = FALSE,
         show_colnames = FALSE,
         ## clustering_distance_rows = "euclidean",
         ## clustering_method = "complete",
         annotation_names_col=F
)

pheatmap(data_cluster3_centrada,
         color=colorRampPalette((brewer.pal(n = 11, name = "PuOr")))(100),
         annotation_colors = colores_grupo,
         annotation_col = grupo_muestras,#anotacion fila
         cutree_rows = 6, #Cortes por grupo
         show_rownames = FALSE,
         show_colnames = FALSE,
         ## clustering_distance_rows = "euclidean",
         ## clustering_method = "complete",
         annotation_names_col=F
)

##Análisis ontológico
```

```
onto_fprf<- read.csv(file.path(results,"onto_fprf.txt"),header=TRUE, sep="\t")
onto_fprf$p.value <- round(onto_fprf$p.value,4)
onto_svmrfe<- read.csv(file.path(results,"onto_svmrfe.txt"),header=TRUE, sep="\t")
onto_svmrfe$p.value <- round(onto_svmrfe$p.value,4)
onto_rfrfe<- read.csv(file.path(results,"onto_rfrfe.txt"),header=TRUE, sep="\t")
onto_rfrfe$p.value <- round(onto_rfrfe$p.value,4)

p_o_fprf <- ggplot(data=onto_fprf, aes(x=Proceso, y=Frecuencia, fill=Tipo)) +
  geom_bar(stat="identity", position=position_dodge())+
  theme_bw()+
  theme(axis.title.y=element_blank(),
        axis.title.x =element_text(size=14),
        axis.text.y = element_text(size=12),
        axis.text.x = element_text(size=10),
        legend.position = c(0.85,0.1),
        legend.title = element_blank(),
        legend.text = element_text(size=12))+
  coord_flip()+
  scale_fill_manual(values=c('#C884D9','#472F4C'))+
  geom_text(aes(label=p.value), position=position_dodge(width=0.9),
              color="white",hjust = 1.3, size=4)

p_o_rfrfe <- ggplot(data=onto_rfrfe, aes(x=Proceso, y=Frecuencia, fill=Tipo)) +
  geom_bar(stat="identity", position=position_dodge())+
  theme_bw()+
  theme(axis.title.y=element_blank(),
        axis.title.x =element_text(size=14),
        axis.text.y = element_text(size=12),
        axis.text.x = element_text(size=10),
        legend.position = c(0.85,0.3),
        legend.title = element_blank(),
        legend.text = element_text(size=12))+
  coord_flip()+
  scale_fill_manual(values=c('#C884D9','#472F4C'))+
  geom_text(aes(label=p.value), position=position_dodge(width=0.9),
            color="white",hjust = 1.3, size=4)

p_o_svmrfe <- ggplot(data=onto_svmrfe, aes(x=Proceso, y=Frecuencia, fill=Tipo)) +
  geom_bar(stat="identity", position=position_dodge())+
  theme_bw()+
  theme(axis.title.y=element_blank(),
        axis.title.x =element_text(size=14),
        axis.text.y = element_text(size=12),
        axis.text.x = element_text(size=10),
        legend.position = c(0.85,0.90),
        legend.title = element_blank(),
        legend.text = element_text(size=12))+
  coord_flip()+
  scale_fill_manual(values=c('#C884D9','#472F4C'))+
  geom_text(aes(label=p.value), position=position_dodge(width=0.9),
            color="white",hjust = 1.3, size=4)
```