FACULTAD DE MATEMÁTICAS

DEPARTAMENTO DE ESTADÍSTICA E INVESTIGACIÓN OPERATIVA

Trabajo Fin de Grado

Machine Learning: aplicación a datos RNA-Seq

Marta Fernández Delgado

Dirigido por:
Dra. María Dolores Cubiles de la Vega
Dra. Alicia Enguix González

Sevilla, Septiembre 2016.

Abstract

Over the past years, biomedicine has experienced a revolution. This is partly due to the very high volume of information existing today that has been produced by both clinical and pharmacological studies as well as Omics data generation. At the same time, the development of statistical and computer techniques that allow its analysis has led to a new area of knowledge: Bioinformatics.

The main aim of this study is to compare different Machine Learning techniques applied to a set of genetic data that were obtained through the RNA-Seq method. This method sequences cDNA in a massive way through a high-performance platform in order to gather global information about the DNA of a sample. Due to the efficacy, reproducibility and performance of the high-throughput sequencing, the RNA-Seq method allows researchers to measure the level of gene expression, detect alternative splicing, mutations, etc.

It is also included in this study the theoretic and practical description of several supervised learning methods such Classification Trees, Bagging, Random Forest and Boosting. The efficacy of these methods has been measured by different rates that can be subsequently compared. The said rates are: sensitivity, specificity, area under the ROC curve, global hit rate, positive predictive value (PPV), negative predictive value (NPV) and balance accuracy.

This study is complemented by the implementation of R software. It presents different graphs (bar charts, box-and-whisker plot, etc.) as well as numerical data of the obtained predictions for each of the methods.

Índice general

Capítulo 1

Introducción a RNA-Seq

1.1. Introducción

En los últimos años se han desarrollado un gran número de proyectos de secuenciación del ADN, como por ejemplo el proyecto Genoma Humano LPGH (1990-2003). En consecuencia, se han perfeccionado y elaborado nuevas técnicas en el campo de la biotecnología. Estas herramientas han permitido estudiar con profundidad los procesos celulares y moleculares, así como comprender mejor el funcionamiento de los sistemas biológicos a nivel molecular. Con el fin de obtener una visión global de los procesos, nació el termino genérico de técnicas ómicas que engloba la genómica, proteómica, metabolómica y transcriptómica.

Todas las técnicas ómicas se basan en el análisis de un gran volumen de datos y conllevan, la comprensión de ciertos mecanismos moleculares y la identificación de biomarcadores o de factores de exposición a ciertas enfermedades.

En nuestro caso, nos vamos a centrar en la transcriptómica, que es la rama que estudia el conjunto de ADN que existe en una célula, tejido u órgano. En concreto, vamos a usar la técnica de RNA-Seq.

La técnica RNA-Seq secuencia de forma masiva el ADNc mediante plataformas de alto rendimiento con el objetivo de obtener información global sobre el ADN de una muestra. Debido a la eficacia, reproductibilidad y rendimiento de la secuenciación de alto rendimiento, la técnica RNA-Seq permite a los investigadores medir el nivel de expresión génica, identificar eventos splicing alternativos, mutaciones, etc.

1.2. Conceptos básicos

En esta sección vamos a definir los conceptos básicos que nos ayudará a comprender la técnica RNA-Seq.

- **Nucleótidos**: Son moléculas orgánicas que constituyen el ADN y ARN. Están compuestos por la unión covalente de una aldopentosa, ribosa (en el ARN) y desoxirribosa (en el ADN), un grupo fosfato unido al carbono 5' de la pentosa y una base nitrogenada. Las bases nitrogenadas pueden ser adenina (A), guanina (G), citosina (C) y timina (T), que en su defecto es el uracilo (U) en el ARN. En la *Figura 1.1* se puede ver la estructura de un nucleótido con las diferentes combinaciones posibles.

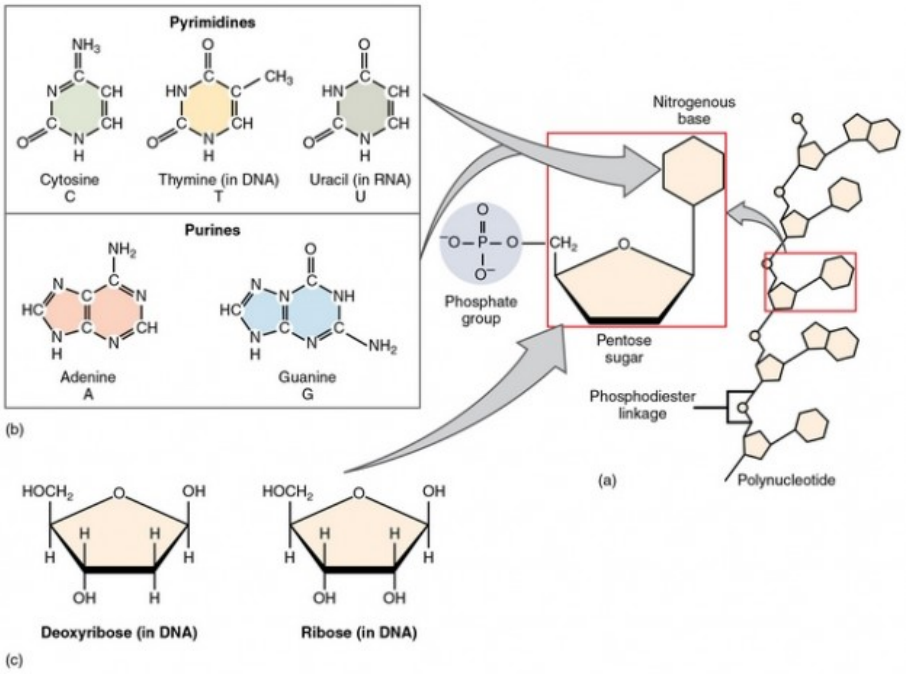


Figura 1.1: Nucleótidos

- **ADN**: Es una doble cadena de desoxirribonucleótidos enrollada en espiral que contiene información genética del individuo. La doble cadena está formada por dos hebras antiparalelas. La base nitrogenada de una condiciona la que se sitúa en su misma posición en la cadena antiparalela. De manera que las parejas son adenina (A) y timina(T), citosina(T) y guanina(G) como se aprecia en la *Figura 1.2*. La forma en la que se distribuyen los nucleótidos es la que codifica la información genética.

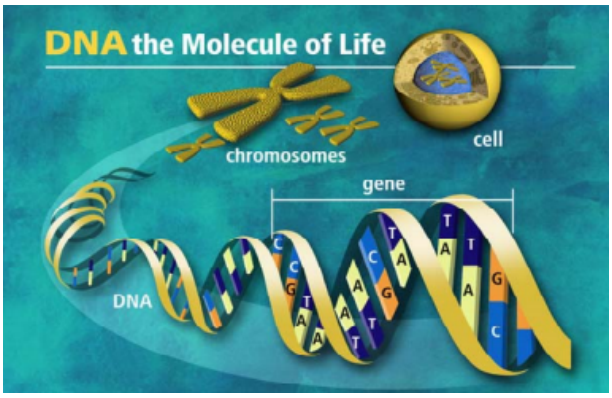


Figura 1.2: Representación del ADN

- **ARNm**: El ARN mensajero es una cadena de nucleótidos (A,C,G,U) donde se encuentra toda la información acerca de la síntesis de aminoácidos de la proteína.

- **ADNc**: Es la hebra del ADN totalmente complementaria al ARNm a partir del cual se ha sintetizado.

- **Gen**: Es una pequeña porción de ADN. Están situados en el mismo lugar del cromosoma en todos los individuos, pero se expresan de manera diferente. Aporta la información necesaria para la síntesis de proteínas. El genoma humano está formado por unos 20.000 genes.
- **Genoma**: Es el conjunto de genes contenido en los cromosomas de los organismos eucariotas. Éstos se encuentran en el núcleo de la célula. Contiene toda la información genética que posee un organismo o una especie.
- **Proteínas**: Es un macromolécula de gran actividad biológica y diversidad funcional constituida por la unión de subunidades llamadas aminoácidos.
- **Enzimas**: Son moléculas de naturaleza proteica cuya función es catalizar reacciones químicas, es decir, acelerar reacciones siempre que sean termodinámicamente posibles.
- **Dogma Central de la Biología Molecular**: El Dogma Central de la Biología Molecular describe el proceso por el cual la información contenida en el ADN se transforma en proteínas. Está formado por dos etapas: transcripción y traducción, como se aprecia en la *Figura 1.3*.
  - *Transcripción*: Proceso en el que la información del ADN es transferida al ARN mensajero.
  - *Traducción*: A partir del ARNm se obtiene la síntesis de proteínas.

  En realidad entre ambos pasos es necesario eliminar ciertas regiones de ARNm que no han sido codificadas, luego no transmiten información.

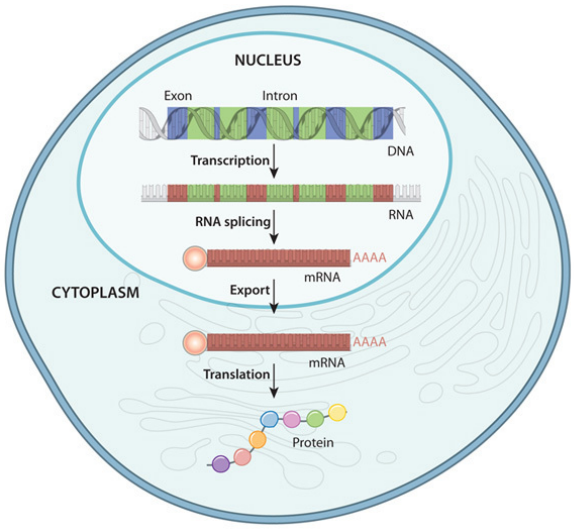


Figura 1.3: Dogma Central de la Biología Molecular

1.3. Proceso histórico

En 1953, Francis Crick y James Watson publicaron en la revista científica Natura su descubrimiento sobre la estructura molecular en forma de doble hélice del ADN, la molécula que porta el programa genético de los organismos vivos.

Después de quince años, se consiguió determinar la primera secuencia de ADN de forma experimental. Las razones de esta demora fueron:

- Las diferentes moléculas de ADN tienen propiedades químicas similares, lo que dificultaba el aislamiento.
- El gran tamaño de la molécula dificultó la secuencia completa.
- Los ADNasas, que son enzimas que catalizan la escisión endonucleolítica de una cadena de ADN específicas, no se conocían. Para secuenciar proteínas se usaba las proteasas que cortaba determinados aminoácidos.

A raíz de que en 1970 se descubriesen unas enzimas capaces de reconocer y fragmentar el ADN en secuencias específicas, se empezaron a desarrollar métodos de secuenciación, es decir, determinar la secuencia de nucleótidos de una muestra de ADN. Frederick Sanger propuso el método "más-menos" (plus-minus method), que consistía en secuenciar fragmentos de ADN usando la enzima ADN polimerasa de E.coli.

Poco después Sanger publica el método que se convertiría en el más utilizado de los siguientes 30 años. Se conoce como método de terminación de cadena de Sanger y fue usado en la secuenciación del genoma humano, debido a su sencillez y precisión.

Posteriormente se desarrolló la técnica de microarrays, que consiste en una superficie sólida a la que se le une una colección de fragmentos de ADN para posteriormente medir el nivel de expresión de cada gen, que se indica generalmente mediante fluorescencia y se mide a través de un análisis de imagen. El esquema de la técnica de microarrays podemos encontrarlo en la *Figura 1.4*.

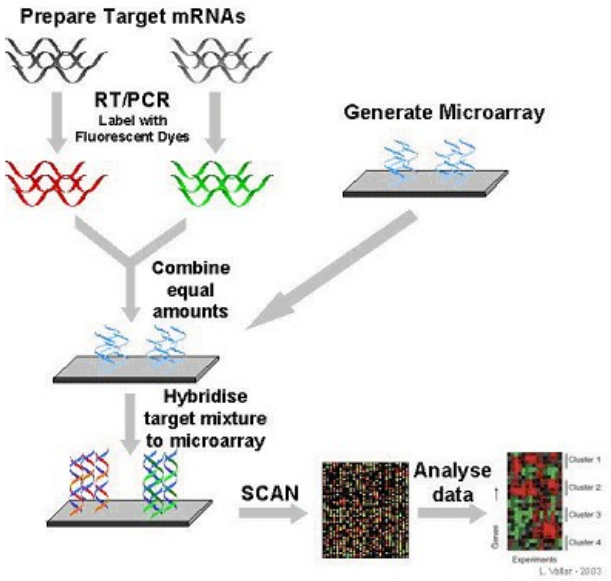


Figura 1.4: Proceso de preparación y obtención de datos de un microarray.

Una de las limitaciones que presenta microarrays es que el rango de detección está limitado debido a la saturación. Para solventar esta deficiencia surgieron las denominadas NGS, secuenciación masiva de nueva generación (Next Generation Sequencing). La técnica RNA-Seq está basada en las tecnologías NGS que explicaremos en el siguiente apartado.

1.4. Procedimiento de obtención de los datos RNA-Seq

La tecnología de secuenciación masiva que vamos a describir es la de Illumina, que es una compañía estadounidense especializada en el ámbito de la Biotecnología.

El proceso de trabajo Illumina se puede resumir en 4 pasos.

1. **Preparación de la genoteca**: Dado un ADN genómico, se fragmenta y se le unen adaptadores a cada fragmento de manera que tienen uno en la dirección 3' y en otro en la dirección 5'. De esta forma ya podemos secuenciar la genoteca como se aprecia en la *Figura 1.5*.

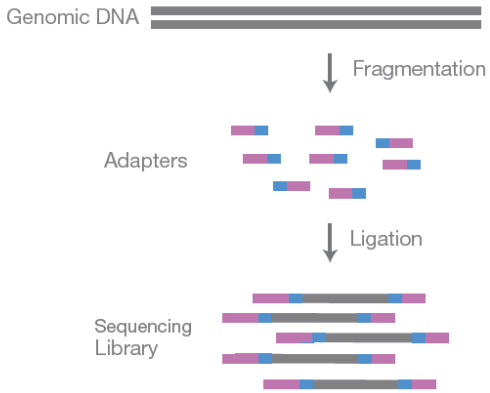


Figura 1.5: Preparación de la genoteca

2. **Amplificación**: Para generar un cluster, se carga la genoteca en una celda de flujo donde los fragmentos se hibridan en la superficie. Se hacen réplicas de cada fragmentos a través de puentes de amplificación, de esta forma se obtienen tantos clusters como fragmentos. El objetivo de hacer esto es para que cada trozo se manifiesten mejor. El proceso de generación de clusters está representado en la *Figura 1.6*.

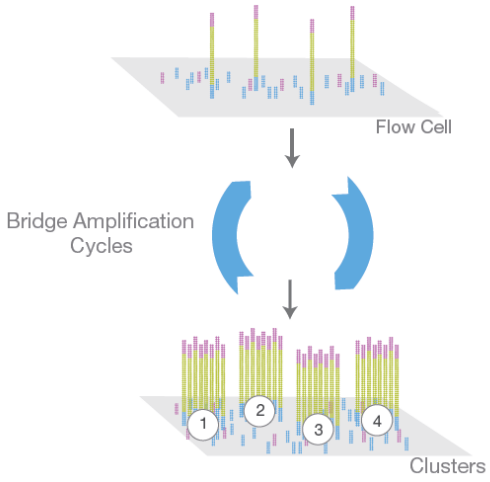


Figura 1.6: Generación de clusters

3. **Secuenciación**: Una vez que se tienen los clusters, el siguiente paso es proceder a secuenciarlos. Para ello se usa reactivos de secuenciación, que incluyen unos niveles de fluorescencia para cada nucleótido. Cuando

la primera base es detectada la celda de flujo capta la imagen y la emisión de cada cluster es grabada. La longitud de ondas y la intensidad son usadas para identificar la base. Este ciclo es repetido n veces para crear una lectura de bases de longitud n. Este proceso se muestra en la *Figura 1.7*.

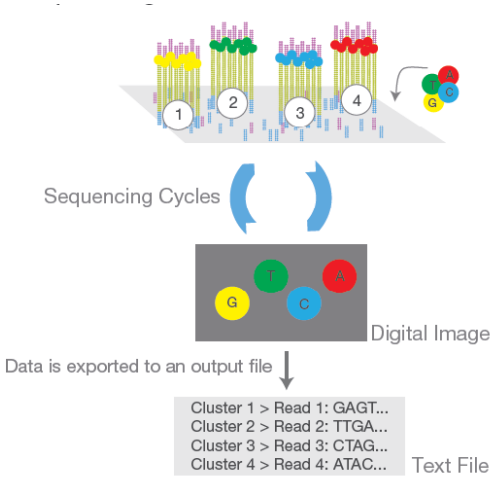


Figura 1.7: Proceso de secuenciación

4. **Alineación y análisis de datos**: Se conoce como *reads* a las lecturas que hemos obtenido en un experimento RNA-Seq, es decir, la secuenciación de aminoácidos que se obtiene al secuenciar cada fragmento de ADN. Estas son alineadas frente a una secuencia de referencia con un software bioinformático. Esto es, se encuentra la posición que ocupa cada read en el genoma de referencia. Después de la alineación las diferencias entre el genoma de referencia y el recién secuenciado puede ser identificado.

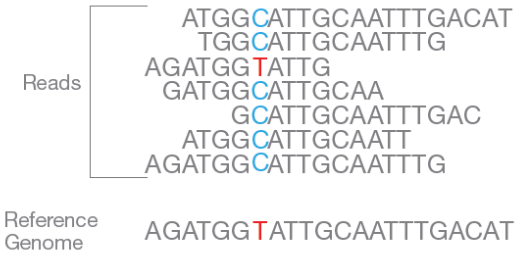


Figura 1.8: Alineación de los *reads*

Hasta ahora lo que hemos explicado es el procedimiento general de la tecnología masiva de nueva generación.

Para aplicar la técnica RNA-Seq, a partir del ARN se obtiene información sobre los genes, de manera que si frente a un gen el número de *reads* es mayor que frente a otro, se dirá que el primer gen muestra un nivel de expresión mayor.

El resumen numérico que obtenemos al aplicar RNA-Seq al análisis de expresión, es decir, al contar los *reads*, son datos de conteo. Estos datos se recogen en tablas, donde las columnas representan los individuos y las filas a cada

uno de los genes. Por ejemplo, podríamos suponer que tenemos dos grupos A y B. Los individuos del grupo A, son aquellos que padecen cierta enfermedad, mientras que los del grupo B son sanos. Cabe destacar que el número de individuos en cada uno de los grupos no tiene por qué ser el mismo. Las filas son los diferentes genes que se le han medido a todos los individuos.

Así, podemos mostrar cuantos conteos tenemos de cada gen en los distintos individuos. Un ejemplo de tabla sería:

	I_{1_A}	$\dots$	I_{n_A}	I_{1_B}	$\dots$	I_{n_B}
Gen 1	$c^1_{1_A}$		$c^n_{1_A}$	$c^1_{1_B}$		$c^1_{n_B}$
$\vdots$						
Gen M	$c^M_{1_A}$		$c^M_{n_A}$	$c^M_{1_B}$		$c^M_{n_B}$

donde I_{i_A} muestra al individuo i-ésimo del grupo A, I_{i_B} es el individuo i-ésimo del grupo B y $c^j_{i_A}, \dots, c^j_{i_B}$ es el dato de conteo para el individuo i-ésimo del grupo A ó B en el gen j-ésimo.

Una vez que tengamos los datos tabulados podemos realizar numerosos estudios que veremos en el siguiente apartado.

1.5. Importancia del análisis de datos de RNA-Seq

Existen numerosas cuestiones biológicas que pueden ser estudiadas desde el punto de vista estadístico. Según el objetivo que tengamos haremos un estudio u otro. A continuación se introducen los análisis de mayor interés.

Comparación de clases
Es uno de los estudios más frecuentes y tiene como objetivo detectar genes diferencialmente expresados. Entendemos que un gen es diferencialmente expresado si para un tejido u organismo su expresión difiere significativamente respecto a los demás genes. Para realizar un experimento, se necesitan diferenciar clases donde se sitúan muestras con individuos con características similares.

Por ejemplo, si queremos hacer una investigación acerca del cáncer cervical, podemos dividir la muestra en pacientes con diferentes grados en el tumor, tratamientos aplicados, etc. Una vez determinadas las clases, procederemos a hacer el análisis cuya finalidad es ver si los perfiles de expresión génica entre las distintas clases difieren, esto es, ver si el nivel medio de expresión es significativamente diferente para las distintas clases.

Predicción de clases
En este caso, también necesitamos tener predefinidas las clases. Sin embargo la finalidad es la búsqueda de una función que nos permita predecir la clase a la que pertenece un nuevo individuo que es a priori desconocido.

En este tipo de estudios la validación y verificación del modelo será muy importante, por ello trataremos con datos cuya clase son conocidas para la validación y datos totalmente independientes para la verificación.

En nuestro trabajo nos centraremos en este estudio.

Descubrimiento de clases
Tenemos como objetivo la identificación de grupos, partimos de datos sin conocer las clases que existen en su

conjunto y utilizando técnicas diversas como por ejemplo el análisis de conglomerados de manera que en cada uno de ellos los individuos sean los más parecidos entre sí y lo más diferente a los demás grupos.

1.6. Normalización

Cuando aplicamos la técnica RNA-Seq, obtenemos el número de fragmentos secuenciados que se mapean frente al genoma de referencia. Los datos que obtenemos se denominan conteos brutos. Debido a la complejidad de la técnica no podemos tener conclusiones fiables, ni realizar comparaciones con esos datos, ya que suelen presentar diferencias considerables en el número total de lecturas. Para solucionar este problema es necesario realizar la normalización.

Existen diversos métodos de normalización, como son: RPKM, TMM, etc.

El método de **RPKM** (Reads Per Kilobase of transcript and Million mapped reads - reads esperadas por kilobase de transcrito por millón de lecturas mapeadas) consiste en normalizar por la longitud de las regiones en cuestión. Se podría considerar como esta región la longitud de los genes.

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

En 2010 Robinson and Oshlack propusieron el método **TMM** (Trimmed Mean of M-values - media truncada de M-valores).

Estos autores defienden que: *"La proporción de lecturas atribuidas a un gen dependen de las propiedades de expresión de la muestra completa y no sólo del nivel de expresión de ese gen".*

Vamos a describir este proceso de manera muy resumida, para ello necesitamos introducir alguna notación.

Y_{gk} Número de conteos observados para el gen g en la librería k.

μ_{gk} Nivel de expresión del gen g en la librería k.

$\mathbf{L}_g$ Longitud del gen g.

$\mathbf{N}_k$ Número total de lecturas en la librería k.

$\mathbf{S}_k$ Salida de RNA total de la muestra.

De manera que el valor esperado de Y_{gk} es:

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

donde

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

El incoveniente de este método es el valor S_k. Se estima a partir de cada muestra, es desconocido y varia considerablemente de una muestra a otra, esto es debido a que depende de la composición de RNA.

Otro método de normalización, que quizás no sea el más sofisticado, pero es bastante común, es el utilizado en el paquete DESeq2 de Bioconductor que consiste en dividir el número de conteos para cada gen de la muestra por el número total de lecturas en dicha muestra. Este método será el que se use en la memoria.

Capítulo 2

Técnicas de Machine Learning

2.1. Árboles de clasificación

2.1.1. Concepto y características de los árboles de clasificación

Sea Y una variable respuesta cualitativa con K clases, modalidades o grupos $C_1, ..., C_K$ y sea $\underline{X} = (X_1, ..., X_p)$, p variables predictoras. El problema consiste en establecer una relación entre Y y $\underline{X}$, es decir, determinar una función objetivo f tal que $Y \simeq f(\underline{X})$ de modo que permita asignar a un caso o individuo una de las clases con el menor error posible. Por tanto, buscamos una regla de clasificación o clasificador.

Para abordar este problema necesitamos un conjunto de datos, que dividiremos aleatoriamente en dos partes: muestra de entrenamiento o aprendizaje y muestra test. La muestra de entrenamiento se usará para entrenar el modelo, es decir, para obtener un método que clasifique a un nuevo individuo en alguna de las modalidades o grupos de la variable respuesta, mientras que la muestra test se usará para estimar la capacidad de generalización del modelo construido.

Para abordar este problema tomamos una muestra de aprendizaje, $\mathfrak{L}$, que viene dada por:

$$\mathfrak{L} = \{(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)\}$$

donde $\underline{x}_i$ representa un elemento de la realización muestral de $\underline{X}$, $y_i \in \{C_1, \ldots, C_K\}$ y n el tamaño muestral de $\mathfrak{L}$.

Entre las variables predictoras $\underline{X}$ pueden aparecer variables tanto cualitativas como cuantitativas.

El método que nos ocupa consiste en realizar una partición del espacio muestral de las variables predictoras a través de la obtención de un conjunto de reglas de decisión. Este conjunto de reglas se puede representar gráficamente mediante un árbol. De ahí, el nombre de estos modelos.

El mecanismo de crecimiento de un árbol de clasificación se realiza mediante un procedimiento de división binaria recursiva, que se explica en apartados posteriores.

De esta forma se tendrá un conjunto de M nodos terminales, asociados a M regiones $\{R_1, \ldots, R_M\}$ que constituyen una partición del espacio predictor, de forma que a cada nodo (o región) se asigna una de las clases con el fin de hacer una predicción para una observación dada.

2.1.2. Notación

Para desarrollar el algoritmo de construcción, se necesita describir la notación que se usará en todo el proceso.

n_t : Número de observaciones en el nodo t.

$n(k)$: Número de observaciones pertenecientes a la clase C_k.

π_k : Probabilidades a priori, es decir, la probabilidad de que una observación pertenezca a la clase k. Pueden ser aportadas por el analista o estimadas como: $\hat{\pi}_k = n(k)/n$.

$n_t(k)$: Número de observaciones de la k-ésima clase, C_k, perteneciente al nodo t.

$v(\underline{X})$: Clase a la que pertenece un caso con valores en las variables predictoras $\underline{X}$.

$d(t)$: Decisión en el nodo t, para todo caso que pertenezca a dicho nodo. Esto es, a todos los casos que pertenezcan al nodo t se les asigna la clase $C_{d(t)}$.

2.1.3. Conceptos asociados al árbol

En este apartado definiremos algunos conceptos relacionados con el árbol de clasificación que nos será de utilidad para el desarrollo del método.

Probabilidad asociada al nodo t: $P[t] = Pr[\underline{X} \in t]$, es la probabilidad de que una observación esté en el nodo t.

Si $\{\pi_k\}$ son conocidas, se estima por:

$$p(t) = \sum_{i=1}^{K} \pi_k \frac{n_t(k)}{n(k)}.$$

Si $\{\pi_k\}$ son desconocidas, se estima por:

$$p(t) = \sum_{i=1}^{K} \frac{n(k)}{n} \frac{n_t(k)}{n(k)} = \sum_{i=1}^{K} \frac{n_t(k)}{n} = \frac{n_t}{n}.$$

Podemos observar que, en este caso, $p(t)$ es la proporción de casos del conjunto de entrenamiento que hay en el nodo t.

Probabilidad de la clase C_k **dado el nodo** t: $P[k|t] = Pr[v(\underline{X}) = k|\underline{X} \in t]$, es decir, la probabilidad de que un individuo pertenezca a la clase C_k sabiendo que está en el nodo t.

Si $\{\pi_k\}$ son conocidas, se estima por:

$$p(k|t) = \frac{\pi_k \frac{n_t(k)}{n(k)}}{\sum_{i=1}^{K} \pi_k \frac{n_t(k)}{n(k)}}.$$

Si $\{\pi_k\}$ son desconocidas, se estima por:

$$p(k|t) = \frac{\frac{n(k)}{n} \frac{n_t(k)}{n(k)}}{\frac{n_t}{n}} = \frac{n_t(k)}{n_t}.$$

Luego, en el último caso, $p(k|t)$ es la proporción de casos de entrenamiento del nodo t que pertenece a la clase C_k.

En general, el clasificador para realizar una división en un nodo viene definida por la clase más probable en ese nodo. Es decir:

Asignar al nodo t a la clase $C_{d(t)}$ para la que se alcanza: $\underset{k=1,\ldots,K}{\text{máx}} p(k|t)$.

Así, se verifica que:

$$p(d(t)|t) = \max_{k=1,\ldots,K} p(k|t).$$

En caso de empate se asignará arbitrariamente.

Una vez que se ha asignado una clase $C_{d(t)}$ a un nodo t, todos los individuos que al evaluar el algoritmo pertenezcan a ese nodo se le asigna la misma clase. Por ello ocurrirá que haya individuos que están en el nodo t y no pertenezcan a la clase asignada a ese nodo. De esta manera estamos cometiendo un error de clasificación.

El error de clasificación que estamos cometiendo en el nodo t se puede medir a través del **estimador por resustitución de la probabilidad de clasificación incorrecta para el nodo** t y se define por:

$$r(t) = 1 - p(d(t)|t).$$

Representa la proporción de individuos que no pertenecen a la clase $C_{d(t)}$ estando en el nodo t.

En general, se define el **riesgo del nodo t** como:

$$R(t) = p(t)r(t).$$

La definición anterior se puede extender al árbol completo, de modo que el **riesgo del árbol** T se define como el **estimador por resustitución de la tasa de error esperada del árbol de clasificación** T, con M nodos finales:

$$R(T) = \sum_{t=1}^{M} p(t)r(t) = \sum_{t=1}^{M} R(t).$$

2.1.4. Elementos necesarios en el algoritmo de construcción

En el apartado anterior se han introducido los conceptos básicos de los árboles de clasificación. A continuación vamos a definir los elementos necesarios para la construcción del árbol.

1. Un conjunto Ω de preguntas binarias para hacer las divisiones.
2. Un criterio para evaluar la bondad de las divisiones.
3. Una regla de parada.
4. Una regla para asignar cada nodo terminal a una clase.

1. Un conjunto Ω de preguntas binarias para hacer la división.

Inicialmente, en el nodo raíz se encuentran las n observaciones de la muestra de aprendizaje. El método consiste en dividir el conjunto de entrenamiento en dos. Para ello se necesita un conjunto de preguntas binarias Ω y seleccionar la más óptima. De esta forma se tienen dos nodos descendientes, uno a la izquierda t_L y otro a la derecha t_R, que en función de la respuesta a la pregunta, nos decantaremos por uno u otro.

Cada una de las divisiones va a depender de una única variable, que en función de su naturaleza, tendrán un número diferente de posibles divisiones, de modo que:

- **Variables binarias**, son del tipo $1/0$, Si/No, etc. Sólo existiría una posible división.

- **Variables cuantitativas u ordinales**, existen infinitas divisiones del tipo:

$$\{x \leq c\} \quad c \in \mathbb{R}.$$

  Aunque en principio existen infinitas divisiones de este tipo, nos podemos quedar con una cantidad finita seleccionando los puntos medios entre dos observaciones ordenadas de la muestra como posibles puntos de la división.

- **Variables nominales** con B modalidades, las divisiones posibles son:

$$[VR_2^B - 2]/2 = [2^B - 2]/2 = 2^{B-1} - 1.$$

Para seleccionar la división óptima en cada nodo la idea es encontar la mejor división. Para cada variable se compararían las p mejores divisiones de las variables individuales y se selecciona la mejor entre todas ellas. Para todo ello será necesario realizarlo según algún criterio que mida la bondad de las divisiones.

2**. Un criterio para evaluar la bondad de las divisiones.**

Situados en el nodo t, una opción para elegir la mejor división sería seleccionar aquella que produce una mayor reducción en el riesgo $R(T)$. Este criterio puede producir árboles de bajo rendimiento, por tener muchos nodos finales. Por ello se introdujo el término de *funciones de impureza*.

Una función impureza Φ se define en el conjunto de todas las K-tuplas de números $(p_1, \ldots, p_K)$ satisfaciendo $p_k \geq 0$, $k = 1, \ldots, K$, $\sum_{k=1}^{K} p_k = 1$, con las propiedades:

$*$ Φ es un máximo sólo en el punto $\left(\frac{1}{K}, \ldots, \frac{1}{K}\right)$.

$*$ Φ logra su mínimo sólo en los puntos $(1, 0, \ldots, 0), (0, 1, \ldots, 0), \ldots, (0, 0, \ldots, 1)$.

$*$ Φ es una función simétrica de $p_1, \ldots, p_K$.

Dada una función de impureza Φ, se define la medida de impureza $I(t)$ de cualquier nodo t como:

$$I(t) = \Phi(P(1|t), P(2|t), \ldots, P(K|t)).$$

Situados en un nodo t, al realizar la división, una parte de los datos se asigna a la rama derecha t_R con probabilidad $P(t_R)$ y otra a la rama izquierda t_L con probabilidad $P(t_L)$, de forma que, la disminución de la impureza en el nodo t es:

$$\Delta I(t) = P(t)I(t) - P(t_R)\ I(t_R) - P(t_L)\ I(t_L) \geq 0.$$

Por tanto, para determinar la división del nodo t, elegiremos la división en la variable que maximice la reducción de la impureza ΔI.

Existen numerosas funciones de impureza, pero las más habituales son el índice de diversidad de Gini y la entropía.

- **Índice de diversidad de Gini**: Dado $(p_1, \ldots, p_K)$ se define el índice de diversidad de Gini como:

$$\phi(p_1, \ldots, p_K) = \sum_{k \neq j} p_k p_j = 1 - \sum_{k=1}^{K} p_k^2.$$

- **Entropía**: Dado $(p_1, \ldots, p_K)$ se define su entropía como:

$$\phi(p_1, \ldots, p_K) = -\sum_{k=1}^{K} p_k \log(p_k).$$

  En el caso en que $p_k = 0$, se considera $0 \log 0 = 0$.

En este momento ya estamos en disposición de construir un árbol de clasificación. Para ello, debemos de seguir los siguientes pasos:

- Determinar el nodo raíz, que incluye todos los individuos del conjunto de entrenamiento.
- Determinamos el par (s, t) donde s es *(variable, división)*, es decir, la variable y el punto por donde se realiza la división y t es el nodo donde la queremos llevar a cabo.
- Aplicamos a cada nodo el paso anterior, hasta que se verifiquen las condiciones de finalización, punto que vemos a continuación.

3**. Una regla de parada**

En el proceso de construcción del árbol necesitamos un criterio que determine una regla de parada. Esta regla marcará las condiciones que se tienen que cumplir para que un nodo no se divida y por tanto sea terminal. En principio podría incluso lograrse un árbol con un nodo final por cada observación, pero en tal caso se obtiene un modelo muy complejo, que aunque tendría $R(T) = 0$, su capacidad de generalización sería baja. En este caso, se dice que el modelo está sobreajustado a los datos.

Algunos criterios de parada que se podrían considerar son:

1. Mínimo número de casos que debe haber en un nodo para intentar dividirlo en dos nodos hijos.
2. Mínimo número de casos en un nodo terminal.
3. Las divisiones deben producir una reducción mínima de la función impureza. Esto es, establecemos un umbral $\beta > 0$, se declara un nodo terminal t si:
$$\max_{s \in S} \Delta I(s, t) < \beta$$
donde S es el conjunto de todos los pares *(variable, división)* que se han realizado en el árbol.

Supongamos que hemos hecho algunas divisiones y hemos llegado a un conjunto de nodos terminales. El conjunto de divisiones usadas, junto al orden en que hemos hecho las divisiones determina lo que llamamos un árbol binario T.

4**. Una regla para asignar una clase a un nodo terminal**

Por último, se requiere una regla de asignación. Para ello se seleccionará una clase que verifique:

$$\text{Si} \quad \underline{x} \in R_t \quad \text{se asigna a la clase} \quad C_{d(t)} \quad \text{para la que se alcanza:} \quad \max_{k=1,\ldots,K} p(k|t).$$

Para evitar los inconvenientes detectados en el método, tales como el excesivo tamaño del árbol que conlleva que se ajusta muy bien a los datos con los que ha sido construído, pero es ineficiente con un conjunto de datos nuevos, se introduce el concepto de poda del árbol con idea de obtener un modelo mejor.

2.1.5. Poda del árbol (Pruning)

El tamaño del árbol (número de nodos terminales) es un parámetro que controla la complejidad del modelo. Un árbol muy pequeño puede que no capture la estructura de los datos y un árbol muy grande puede sobreajustarse a los datos.

Una vez construído un árbol T_0, según se describió en el apartado anterior, se procede a la poda del mismo, cortando sucesivamente ramas o nodos terminales que representen poco aporte a la explicación de la variable respuesta,

encontrando así el tamaño adecuado del árbol.

En primer lugar, hay que construir un árbol lo mayor posible T_0 permitiendo que el criterio de división continúe hasta que la regla de parada del criterio de crecimiento del árbol determine cuáles son los nodos terminales.

Luego procederemos a la poda que parte de un nodo t. Consiste en eliminar de T_0 todos los descendientes de dicho nodo, esto es, eliminar todo lo que esté por debajo del nodo t, tal y, como vemos en la *Figura 2.1*; el árbol original T_0 es el de la izquierda y el podado T el de la derecha. Si llegamos a un subárbol T a partir de T_0 por sucesivas podas de ramas, entonces T es llamado subárbol podado de T_0 y se denota por $T \subseteq T_0$. Tengamos en cuenta que T_0 y T tienen el mismo nodo raíz.

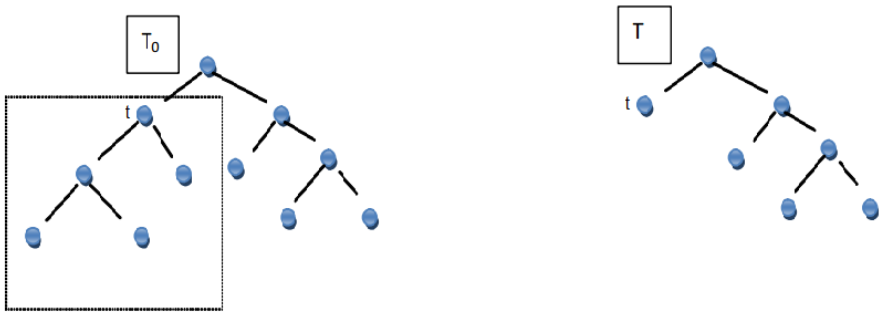


Figura 2.1: Poda del árbol

Secuencia de subárboles

Debido a que un árbol se puede podar por cada uno de sus nodos de forma anidada, existe una selección de subárboles con tamaño cada vez más pequeños. Una vez creada la secuencia tenemos que seleccionar cuál es el mejor subárbol.

Para ello definimos el **Criterio de Coste-Complejidad**. Consiste en considerar una secuencia de árboles indexada por un parámetro de ajuste α no negativo. A cada α le corresponde un árbol $T_\alpha \subseteq T_0$ tal que:

$$C(T_\alpha, \alpha) = \min_{T \subseteq T_0} C(T, \alpha) \quad \text{siendo} \quad C(T, \alpha) = R(T) + \alpha|T|.$$

donde $|T|$ es el tamaño de T y $R(T)$ el riesgo del árbol que podría sustituirse por la función impureza según se use para la construcción del árbol.

El parámetro α controla el compromiso entre el tamaño del árbol y el ajuste a los valores dados. Valores altos de α conducen a un tamaño de árbol pequeño y viceversa. Para cada valor de α, encontramos subárboles $T_\alpha \subseteq T_0$ que minimizan $C(T, \alpha)$. Cuando $\alpha = 0$, entonces el subárbol T_α será igual a T_0, pues el árbol coincide con el inicial.

A pesar de que los valores de α son infinitos, el número de subárboles es finito que denotaremos por m. Esto es debido a que para el intervalo $[\alpha_h, \alpha_{h+1})$ el árbol óptimo es el mismo. De hecho, se puede obtener una familia anidada de subárboles $\{T^{(h)}\}_h$ y una serie de valores $\alpha_1 < \ldots < \alpha_{m-1}$ tales que $T^{(h)} = T_\alpha$ para todo $\alpha \in [\alpha_h, \alpha_{h+1})$, $h = 1, \ldots, m-2$.

Selección del árbol óptimo

Una vez generada la secuencia de subárboles, debe elegirse uno de ellos. Luego, se debe disponer de un estimador insesgado del error esperado de cada subárbol.

El método anterior crea una secuencia decreciente de subárboles anidados $T_0 \supseteq T_2 \supseteq \ldots \supseteq t_1$, donde t_1 es el nodo raíz, es decir, el mínimo subárbol posible de T_0. El siguiente problema que se nos plantea es la selección del árbol óptimo.

Tenemos que seleccionar un árbol de la secuencia que será el que finalmente usaremos para futuros estudios. Para ello se asocia una medida de error a cada árbol y se elige aquél que tenga asociado un menor error.

$$\text{Elegimos} \quad T^{(k_0)} \quad : \quad C(T^{(k_0)}) = \min_k C(T^{(k)}) \quad \text{con} \quad C(T) = C(T, 0).$$

Luego, se debe disponer de un estimador insesgado del error (riesgo) esperado de cada subárbol. Este estimador se obtiene mediante validación cruzada.

Para construir el estimador mediante validación cruzada de k iteraciones (o k-fold cross-validation), $C_{cv}(T)$, se divide la muestra de aprendizaje $\mathfrak{L}$ en k subconjuntos. Generalmente se considera el número de iteraciones $k = 10$, de modo que, para la primera iteración, se entrena un modelo para $k-1$ subconjuntos, dejando fuera el primero, para posteriormente evaluarlo. Este proceso se lleva a cabo k veces. En la *Figura 2.2* se representa lo citado anteriormente. Representa un proceso de validación cruzada de 10 iteraciones, en cada una de ellas el conjunto test (azul) va cambiando y se realiza una estimación en cada caso. Finalmente, se hace una estimación final ponderando cada una de las estimaciones.

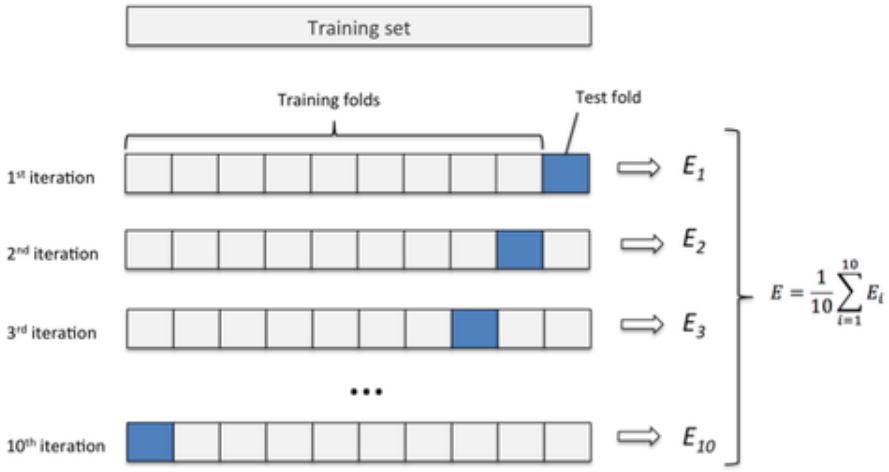


Figura 2.2: Esquema de validación cruzada

Una vez que tenemos cada uno de los subconjuntos evaluados para las diferentes iteraciones, se calcula la media de las k medidas obtenidas a través de las iteraciones, que denotaremos como $C_{cv}(T)$. Normalmente nos decantamos por este método cuando la muestra no es lo suficientemente grande.

A pesar de las mejoras que le hemos hecho, este algoritmo es muy inestable, una pequeña variación en el conjunto de datos conlleva un árbol totalmente distinto.

El estimador visto de $C(T)$ en función del número de nodos terminales $|T^{(k)}|$, se comporta, como vemos en la *Figura 2.3*, de la siguiente forma: un decrecimiento inicial, luego existe una zona relativamente constante, y por último crece para valores más elevados de $|T^{(k)}|$.

El método de validación cruzada tiene un riesgo debido a que la estimación se hace en función del conjunto de entrenamiento. Para evitar la inestabilidad del estimador es recomendable calibrar la incertidumbre de $C_{cv}(T)$ calculando su error estándar (SE). El error estándar asociado al estimador $C_{cv}(T)$, viene dado por:

$$SE(C_{cv}(T)) = \sqrt{C_{cv}(T)\frac{1 - C_{cv}(T)}{|\mathfrak{L}_T|}}.$$

donde $\mathfrak{L}_T$ es el subconjunto de los datos de la muestra test que se han usado para construir el árbol T.

Como vemos en la *Figura 2.3*, el mínimo es muy inestable. De ahí surge la idea de $1 - SE$ (1 - error estándar) como alternativa a la selección del mínimo con el objetivo de reducir esta inestabilidad a la vez que nos aseguramos que el árbol seleccionado sea el más simple con valor cercano al mínimo. En la *Figura 2.4* vemos que el mínimo de $C_{cv}(T)$ se alcanza en k_0. De ahí calculamos $SE(C_{cv}(T^{(k_0)}))$ y seleccionamos aquel punto que esté por debajo de la recta de SE con el mínimo número de nodos posibles, en este caso es k_1.

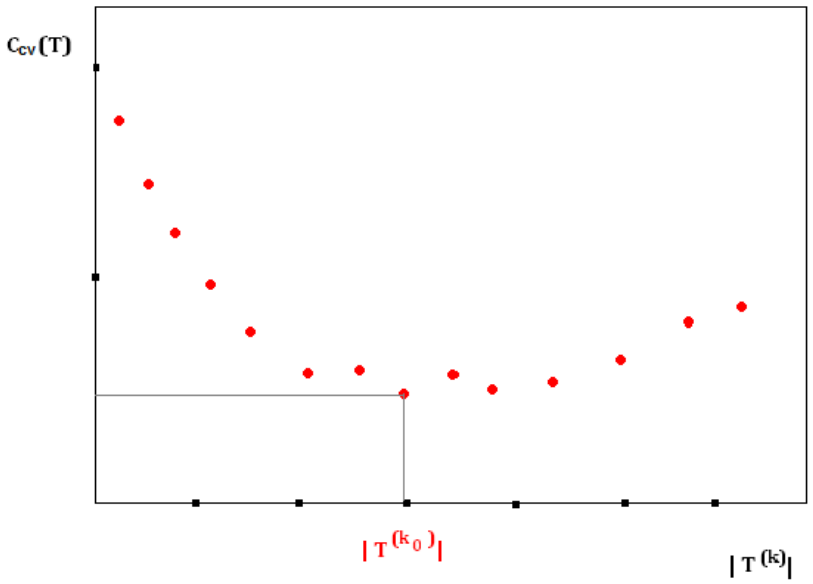


Figura 2.3: Estimador de $C_{cv}(T)$ en función de los nodos terminales.

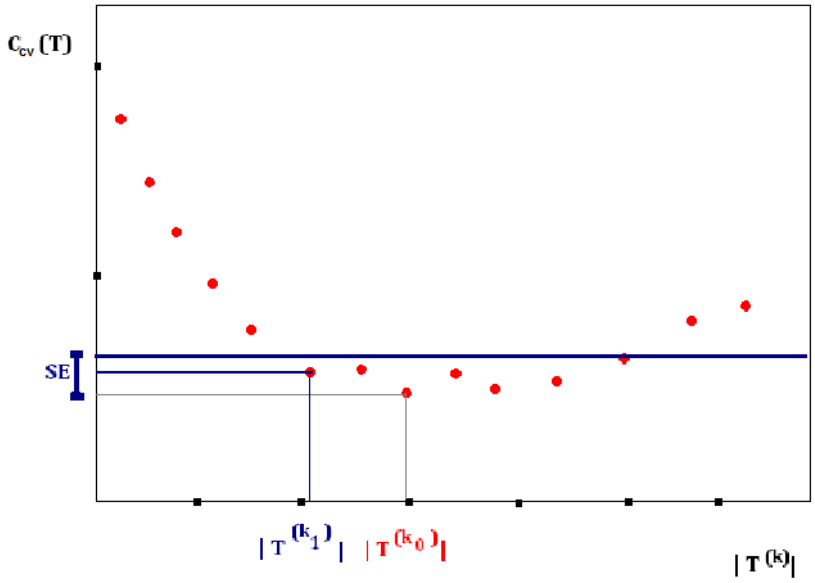


Figura 2.4: Estimador de $C_{cv}(T)$ en función de los nodos terminales con el error estándar SE.

De modo que la regla $1-SE$ se formula de la siguiente manera:

$$\text{Si} \quad T^{(k_0)} \quad : \quad C_{cv}(T^{(k_0)}) = \min_k C_{cv}(T^{(k)}) \quad \text{entonces seleccionamos} \quad T^{(k_1)} \quad \text{con} \quad k_1 \quad \text{tal que:}$$

$$k_1 = \text{máx}\{k : C_{cv}(T^{(k)}) \leq C_{cv}(T^{(k_0)}) + SE(C_{cv}(T))\}.$$

Por tanto, se selecciona el árbol más simple, que será el que tenga menos nodos terminales (el árbol de mayor subíndice) y que no se desvía más de $1-SE$ del mínimo.

Una manera de que este problema de clasificación tenga mejor rendimiento es hacer una agregación de modelos. Esto es, a partir de una muestra de aprendizaje $\mathfrak{L}$, construir varios predictores y los combinamos de manera que obtenemos una predicción más estable, con mayor rendimiento y disminuyendo la varianza.

En función de los modelos intermedios y de las formas de combinarlos, podemos considerar los siguientes algoritmos de agregación de modelos: Bagging, Random Forest y Boosting que se estudiarán en las secciones siguientes.

2.1.6. Ventajas y desventajas de los árboles

Una vez visto el procedimiento de construcción del árbol describimos algunas ventajas e incovenientes de este modelo:
Ventajas:

- Puede ser aplicado para cualquier estructura de datos a través de una formulación apropiada del conjunto de cuestiones Ω.
- La clasificación final tiene una forma simple que puede ser almacenada de manera compacta y clasifica eficientemente nuevos datos.

- La selección de variables se hace paso a paso, automático y reducciendo el coste de complejidad. Se busca nodo a nodo hasta conseguir la división más significativa. En cada etapa se intenta extraer la información más relevante de la parte del espacio que se está trabajando.
- Nos aporta no solo una clasificación si no también una estimación de la probabilidad de clasificar un objeto erróneamente.
- Una estructura de datos estandarizados, es invariante bajo transformaciones monótonas de las variables continuas.
- Es muy robusto respecto a los outliers y los puntos mal clasificados en L.
- Es muy fácil de interpretar y explicar.
- Los árboles de decisión toma decisiones muy cercanas a las que tomaría un humano.
- Se pueden visualizar gráficamente.
- Pueden manejar fácilmente predictores cualitativos sin la necesidad de crear variables ficticias y las posibles interacciones se incluyen automáticamente.
- En conjunto de datos grandes puede revelar estructuras complejas.
- Es una metodología no paramétrica.

Desventajas:

- Clasifica de manera aleatoria cuando tenemos valores perdidos.
- Aunque la optimalidad se aplique a cada división, esto no significa que el árbol sea óptimo.
- Las variables predictoras continuas han de ser discretizadas.
- Es posible que las interacciones débiles se impongan a las más fuertes.
- Los árboles grandes tienen tendencias a sobreajustar a los datos.
- Son inestables, es decir, pequeños cambios en los datos iniciales pueden producir árboles muy distintos.

2.2. Bagging

A la hora de buscar una regla de clasificación para un conjunto de datos, uno de los problemas más frecuentes es el tamaño limitado de la muestra. Cuanto más pequeña sea la muestra de aprendizaje, más difícil será que represente adecuadamente la población total. Suele ocurrir que las clasificaciones obtenidas con un conjunto pequeño sean inestables, es decir, pequeños cambios en los datos iniciales pueden producir árboles muy distintos. En estos casos el clasificador no funcionará de manera eficiente. Actualmente existen numerosas técnicas que intentan solventar este problema para así poder tener clasificadores más estables.

En este punto nos centraremos en la técnica Bagging, que fue propuesto por Breiman en 1996. Su nombre procede de **B**ootstrap **agg**rega**ting** y se basa en los métodos bootstrap y de agregación. En esta memoria vamos a usar Bagging para la clasificación binaria, pero es posible aplicarlo para más de una clase.

Dada una muestra de aprendizaje $\mathfrak{L} = \{(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)\}$, el método de bootstrap consiste en obtener B muestras aleatorias con reemplazamiento de igual tamaño que el original. Para ello se asigna a cada una de las n observaciones el mismo peso $1/n$. Se muestrea de forma aleatoria y con reemplazamiento obteniendo B nuevas muestras $\mathfrak{L}_1^*, \ldots, \mathfrak{L}_B^*$ siendo la distribución asintótica de cada una de ellas próxima a la empírica de la muestra

original.

En particular, para aplicar Bagging realizamos B muestras bootstrap $\mathfrak{L}_1^*, \ldots, \mathfrak{L}_B^*$ del conjunto de entrenamiento $\mathfrak{L}$. Para cada uno de estos conjuntos se construye un clasificador simple con el que se evaluará a los individuos de la muestra original que no estén en el conjunto con el que se han construido el clasificador, estas observaciones se conocen como out-of-bag (OOB).

Una vez construido los B clasificadores se combinan usando la clase de la variable respuesta y más predicha. De esta forma se crea el clasificador combinado:

$$C(\underline{x}) = Arg\,\underset{y}{\text{máx}} \sum_{b:C_b(\underline{x})=y} 1.$$

donde C_b es el clasificador de la $b-$ésima muestra bootstrap. Es decir, $C(\underline{x})$ es la clase más frecuente.
El siguiente algoritmo muestra de manera esquemática el método de Bagging que podemos apreciar en la *Figura 2.5*.

Algoritmo 1 Bagging para clasificación

Entrada: Sea $\mathfrak{L} = \{(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)\}$ donde $\underline{x}_i$ representa un elemento de la realización muestral de $\underline{X}$, $y_i \in \{-1, 1\}$.

1: Tomamos $b = 1, \ldots, B$:

1. Se construye la muestra bootstrap b-ésima, $\mathfrak{L}_b^*$ de tamaño n.
2. Se entrena un árbol T_b, utlizando $\mathfrak{L}_b^*$, y se calcula el clasificador: $C_b(\underline{x})$.

2: El estimador Bagging es:

$$C(\underline{x}) = Arg\,\underset{y}{\text{máx}} \sum_{b:C_b(\underline{x})=y} 1.$$

A la hora de extraer una muestra bootstrap del conjunto de entrenamiento $\mathfrak{L} = \{(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)\}$, la probabilidad de que la observación $\underline{x}_i, i = 1, \ldots, n$, aparezca m veces $(m = 0, 1, \ldots, n)$ en la muestra $\mathfrak{L}_b$, viene dada por una distribución binomial $B(n, 1/n)$ donde $1/n$ es la probabilidad de ser seleccionado en cada extracción y n es el número de extracciones con reposición que se efectuan. Luego, la probabilidad de que $\underline{x}_i$ sea extraída m veces será:

$$P(m) = \binom{n}{m} \left(\frac{1}{n}\right)^m \left(1 - \frac{1}{n}\right)^{n-m}$$

En el caso de que $1/n < 0{,}1$, es decir, cuando el número de observaciones es mayor de 10, se puede aproximar por una distribución de Poisson de parámetro $\lambda = n\frac{1}{n} = 1$. De esta forma, la probabilidad de que $\underline{x}_i$ sea extraída m veces es:

$$P(m) \simeq \frac{e^{-1}}{m!}$$

Luego, la probabilidad de que una observación no esté incluida en la muestra bootstrap es $P(0) = 1/e = 0{,}3678$. En media, el 37 % de las observaciones se quedará fuera de la muestra de bootstrap, es decir, estarán en OOB. Así podrá ocurrir que las posibles observaciones ruidosa del conjunto no hayan sido seleccionadas en algunas de las muestras. Cuando ocurra esto, el clasificador construido será mejor que el del conjunto de entrenamiento original con observaciones ruidosas. Estos clasificadores deberán tener una mayor influencia en la decisión final. Esto explica que el clasificador obtenido por Bagging obtenga mejores resultados que los clasificadores individuales.

Los métodos que utiliza un clasificador final como combinación de clasificadores individuales calculados en el método es frecuente que obtengan mejores resultados. Esto puede entenderse si se considera que al unir los clasificadores individuales se están aumentando las ventajas de cada uno de ellos en el clasificador final.

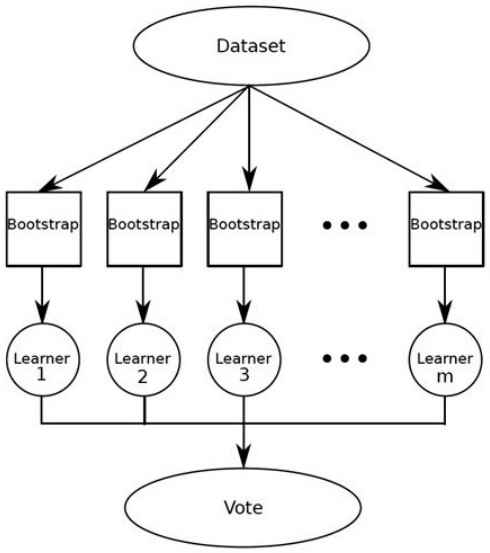


Figura 2.5: Esquema del procedimiento del método Bagging

Según Breiman, el método de bagging es óptimo para clasificadores inestables debido a que logra reducir la variabilidad, mientras que para procedimientos estables pueden llegar incluso a perjudicar el comportamiento del clasificador.

Como desventajas, una de las principales es la pérdida de interpretación de los resultados debido a que no es posible expresar el resultado mediante un árbol. Como consecuencia, ahora no tenemos información de qué variables son más importantes en el procedimiento. Por tanto, mejoramos la precisión de la predicción a costa de la pérdida de la interpretabilidad.

2.3. Random Forest

Breiman en 2001 propone el algoritmo Random Forests (bosques aleatorios) con la idea de reducir la variabilidad de Bagging y disminuir la correlación entre los distintos árboles. Esto se lleva a cabo en el proceso de crecimiento del árbol a través de la selección aleatoria de las variables de entradas.

Para iniciar el método de Random Forest consideramos una muestra de aprendizaje $\mathfrak{L} = \{(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)\}$.

Se genera aleatoriamente una muestra boostrap. Como se vió en el capítulo anterior, en este tipo de muestra la probabilidad de que un individuo no esté en el conjunto de entrenamiento es aproxidamente $37\,\%$. Luego tenemos dos conjuntos: el conjunto de entrenamiento generado por la muestra bootstrap y el conjunto de validación con los individuos out-of-bag (OOB).

En cada nodo, se busca la mejor variable entre un subconjunto aleatorio de variables de tamaño m, $m \leq p$ para dividir los datos. El método para seleccionar la mejor división será el mismo que el aplicado en la sección de Árboles de Clasificación. Se aconseja que el valor de m sea $\sqrt{p}$. La elección de este subconjunto es aleatorio. A continuación, se busca la mejor división de los datos de entrenamiento teniendo en cuenta solo las m variables seleccionadas. Para ello se implementa una función objetivo, que normalmente será la función de impureza, el índice de Gini o entropía.

Los procesos anteriores son repetidos tantas veces como se considere oportuno, de forma que se tiene un conjunto de árboles de decisión entrenados sobre diferentes conjuntos de datos y atributos. Una vez entrenado el método, la evaluación del conjunto test se realiza por el conjunto de árboles generados. La categoría final de la clase se determina por la clase más frecuente teniendo en cuenta el conjunto de árboles. Cabe destacar que cada registro de la muestra de aprendizaje habrá veces que sea usado para construir el clasificador y otras veces estará en la muestra de validación.

Para medir el error del algoritmo se usar el error OOB, es decir, para cada árbol generado usamos la muestra no seleccionada en el bootstrap de entrenamiento para clasificarlas con dicho árbol. Promediando sobre el conjunto de árboles podemos estimar el error total del algorítmo.

Supongamos que hay una variable que explica muy bien la variable respuesta, pero hay otras que la explican moderadamente bien. Al aplicar el método de Bagging, la gran mayoría de los árboles utilizará las variables que la explica de manera muy fuerte para hacer la primera división. Y posteriormente, las variables que son moderadamente fuertes. Esto justifica el sentido de seleccionar un subconjunto de m variables predictivas.

En consecuencia, los árboles serán muy similares entre sí, luego las predicciones de los árboles estarán fuertemente correladas. Esto conlleva a que la predicción final será un promedio de muchas cantidades altamente correladas que desafortunadamente no conducen a una reducción considerable de la variabilidad, como ocurre cuando promediamos muchas cantidades no correladas.

La selección de un número pequeño de m variables predictoras será útil en la construcción de un bosque aleatorio cuando se tiene un gran número de variables predictoras correladas entre sí.

A continuación se muestra un resumen del algoritmo.

Algoritmo 2 Random Forest para clasificación

Entrada: Dada una muestra de aprendizaje: $\mathfrak{L} = \{(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)\}$ donde $\underline{x}_i$ representa un elemento de la realización muestral de $\underline{X}$, $y_i \in \{C_1, \ldots, C_K\}$.

1: Tomamos $b = 1, \ldots, B$:

1. Seleccionamos la muestra bootstrap b-ésima de tamaño n del conjunto de entrenamiento.
2. Hacemos crecer el árbol de random forest T_b para los datos de la muestra bootstrap, se repiten los siguientes pasos hasta que se verifiquen algunos de los criterios de parada vistos en la sección 1.
   - *a*) Seleccionamos m variables al azar de las p variables.
   - *b*) Seleccionamos la mejor variable/punto de división entre las m.
   - *c*) Dividimos el nodo en dos nodos hijos.
3. Devuelve el conjunto de los B árboles generados.

2: Hacemos la predicción un nuevo individuo $\underline{x}$:
Sea $C_b(x)$ la clase de la predicción del b-ésimo árbol de random forest. Entonces:

$$C(\underline{x}) = Arg\,\underset{y}{\text{máx}} \sum_{b:C_b(\underline{x})=y} 1.$$

2.4. Boosting

Al igual que en los métodos anteriores se necesita una muestra de aprendizaje $\mathfrak{L}$, para que al aplicar el algoritmo seamos capaces de encontrar un clasificador capaz de predecir la clase de una nueva observación. La precisión de la regla de clasificación dependerá de la calidad del método empleado y de la aplicación en concreto.

Boosting, a diferencia de los métodos anteriores, hace crecer árboles secuencialmente, es decir, cada árbol se cosntruye usando información procendente de los árboles anteriores. Este método consiste en determinar una regla de clasificación débil, que es una regla que funciona ligeramente mejor que una clasificación al azar, minimizando la suma de los pesos de los individuos mal clasificados.

De esta forma se consigue que para la siguiente iteración el modelo otorgue más relevancia, es decir, más peso a los individuos mal clasificados por los modelos anteriores.

El algoritmo de Boosting funciona de la siguiente manera:

- Partimos de un algoritmo base que se aplica en cada iteración sobre el conjunto de entrenamiento reponderado, de modo que se produce una secuencia de clasificadores débiles $C_m(\underline{x}), m = 1, 2, \ldots, M$.
- La modificación de los datos en cada iteración depende del resultado del clasificador anterior. En los datos utilizados para generar el clasificador C_m se da mayor peso a las observaciones clasificadas erróneamente por el clasificador C_{m-1}.
- Los M clasificadores se combinan a través de una ponderación, donde el peso asignado a cada clasificador depende de su tasa de error.

Existen numerosos métodos de Boosting que se puede clasificar según la naturaleza de la función de pérdida o agregación. Algunas autores proponen una función de pérdida del tipo exponencial, como usa el algoritmo Adaboost. Sin embargo, otros autores apoyan que para clasificación, la función de pérdida se debe elegir como el logaritmo de verosimilitud binomial, es el caso del método Logitboost.

En esta sección se va a aplicar el método de Boosting a un conjunto de datos con variable respuesta dicotómica, para ello nos vamos a centrar en desarrollar LogitBoost para la clasificación binaria. Este método será el que usaremos en la aplicación práctica.

Algoritmo Logitboost

Friedman desarrolló LogitBoost en el año 2000. Este método utiliza los árboles de decisión como modelos de base, y originalmente fue pensado para la clasificación binaria.

Se parte de una muestra de aprendizaje $\mathfrak{L} = \{(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)\}$ con $\underline{x}_i$ representa un elemento de la realización muestral de $\underline{X}$ e $y_i \in \{-1, 1\}$, y se intenta minimizar una función de agregación. Los valores $\{-1, 1\}$ son una codificación de los valores de la variable respuesta.

El algoritmo Logitboost se basa en seleccionar la función de agregación como el logaritmo de la verosimilitud binomial. La filosofía es la misma, es decir, entrenar un determinado número de veces el clasificador asignando más peso a los individuos mal clasificados por los modelos anteriores.

Se inicia el método asignando a la función de agregación el valor 0, es decir, $F^{(0)}(\underline{x}) = 0$, y la probabilidad de seleccionar la clase -1, $1/2$, esto es $p^{(0)}(\underline{x}) = 1/2$.

La probabilidad de que la variable respuesta fuese uno para el individuo i-ésimo en la iteración $m - 1$ se denota por $p^{(m-1)}(\underline{x}_i)$, se calcula como:

$$p^{(m-1)}(\underline{x}_i) = P_w(Y = 1/\underline{X} = \underline{x}_i)$$

donde el subíndice w indica que la probabilidad es calculada sobre el conjunto ponderado utilizando los pesos w_i correspondientes a cada iteración y m es la iteración.

Se realizan m iteraciones, para cada una de ellas se calcula la variable respuesta para cada uno de los individuos de la siguiente manera:

$$z_i^{(m)} = \frac{y_i - p^{(m-1)}(\underline{x}_i)}{w_i^{(m)}}$$

donde $w_i^{(m)}$ es el peso de la iteración m-ésima para el individuo i-ésimo y se calcula $w_i^{(m)} = p^{(m-1)}(\underline{x}_i)(1 - p^{(m-1)}(\underline{x}_i))$.

A continuación ajustamos la función de pérdida o agregación por el método de los mínimos cuadrados ponderados, de manera que la expresión es la siguiente:

$$f^{(m)} = Argmin_f \sum_{i=1}^{n} w_i^{(m)}(z_i^{(m)} - f(\underline{x}_i))^2.$$

Esta función es usada para recalcular la función de agregación, cuya expresión es:

$$F^{(m)}(\underline{x}_i) = F^{(m-1)}(\underline{x}_i) + \frac{1}{2} f^{(m)}(\underline{x}_i).$$

Se actualizan las probabilidades $p^{(m)}(\underline{x}_i)$ como:

$$p^{(m)}(\underline{x}_i) = (1 + \exp(-2F^{(m)}(\underline{x}_i)))^{-1}.$$

Por último, se calcula el clasificador final de la siguiente manera:

$$C^{(m)}(\underline{x}_i) = sign(F^{(m)}(\underline{x}_i)).$$

Cabe destacar que si despejamos $F^{(m)}(\underline{x}_i)$ de la función que calcula las probabilidades $p^{(m)}(\underline{x}_i)$, nos queda la expresión del logaritmo de la verosimilitud de la siguiente manera:

$$F(\underline{x}) = \frac{1}{2} \log\left(\frac{p(\underline{x})}{1 - p(\underline{x})}\right).$$

De forma más esquemática podemos resumir el algoritmo de la siguiente manera:

Algoritmo 3 Logitboost

Entrada: Sea $L = \{(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)\}$ donde $\underline{x}_i$ representa un elemento de la realización muestral de $\underline{X}$, $y_i \in \{-1, 1\}$

1: Se inicializa una función de agregación $F^{(0)}(\underline{x}) = 0$ y probabilidades $p^{(0)}(\underline{x}) = \frac{1}{2}$.

2: Para $m = 1, \ldots, M$:

1. Calcular los pesos y la respuesta para $i = 1, \ldots, n$.

$$w_i^{(m)} = p^{(m-1)}(\underline{x}_i)(1 - p^{(m-1)}(\underline{x}_i)) \quad ; \quad z_i^{(m)} = \frac{y_i - p^{(m-1)}(\underline{x}_i)}{w_i^{(m)}}.$$

2. Se ajusta una función $f^{(m)}$ por mínimos cuadrados ponderados:

$$f^{(m)} = Argmin_f \sum_{i=1}^{n} w_i^{(m)} (z_i^{(m)} - f(\underline{x}_i))^2.$$

3. Actualizamos la función de agregación, el clasificador y calculamos las nuevas probabilidades:

$$F^{(m)}(\underline{x}_i) = F^{(m-1)}(\underline{x}_i) + \frac{1}{2} f^{(m)}(\underline{x}_i)$$

$$C^{(m)}(\underline{x}_i) = sign(F^{(m)}(\underline{x}_i))$$

$$p^{(m)}(\underline{x}_i) = (1 + \exp(-2F^{(m)}(\underline{x}_i)))^{-1}.$$

Capítulo 3

Aplicación a datos RNA-Seq

En esta sección vamos a aplicar los diferentes métodos de clasificación vistos en teoría sobre un conjunto de datos. Los datos que vamos a usar se encuentran en el paquete de Bioconductor `MLSeq`. Dicho conjunto denominado Cervical procede de un estudio realizado por el Centro Nacional para la Información Biotecnológica (NCBI) que es parte de la Biblioteca Nacional de Medicina de Estados Unidos. Contiene 714 expresiones de genes de miRNA de una muestra de 58 humanos, donde se diferencian dos clases de individuos, 29 de ellos que han desarollado un tumor cervical ("T", tumor) y 29 que no ("N", no tumor).

Para medir la eficiencia de los métodos y poder realizar una comparación entre ellos necesitamos calcular diferentes tasas: sensibilidad, especificidad, área bajo la curva ROC, tasa de acierto global, valor predictivo positivo (PPV), valor predictivo negativo (NPV) y balance accuracy. A continuación definimos las diferentes tasas, para ello necesitamos tener presente la siguiente tabla:

	Positivos reales (T)	Negativos reales(N)
Predicción Positivos	Verdaderos Positivos (VP)	Falsos Positivos (FP)
Predicción Negativos	Falsos Negativos (FN)	Verdaderos Negativos (VN)

Se entenderá como positivo tener cáncer cervical, es decir, el individuo tiene T en la variable respuesta, mientras que si es negativo será clasificado como N.

- **Sensibilidad**: Se define como la proporción de los individuos que han sido clasificados como positivos y que realmente son positivos (VP). Su expresión es:

$$Sensibilidad = \frac{VP}{VP + FN}.$$

  El valor de la sensibilidad oscila entre 0 y 1. De modo que cuanto más próximo esté a 1, el clasificador será capaz de detectar mejor los positivos (enfermos) a través del método que se esté haciendo.

- **Especficidad**: Es la proporción de individuos que han sido clasificados como negativos y que realmente son negativos (VN). Se expresa como:

$$Especificidad = \frac{VN}{VN + FP}.$$

  Nos indica la proporción de individuos que somos capaces de identificar como sanos a través de la prueba.

  Al igual que en el caso anterior, el valor oscila entre 0 y 1, cuanto más proximo esté de 1, mejor será el método para detectar a individuos negativos, es decir, los sanos.

- **Tasa global de aciertos**: Informa de la proporción de casos bien clasificados, ya sean positivos o negativos.

$$Tasa \quad global = \frac{VP + VN}{VP + FP + VN + FN}.$$

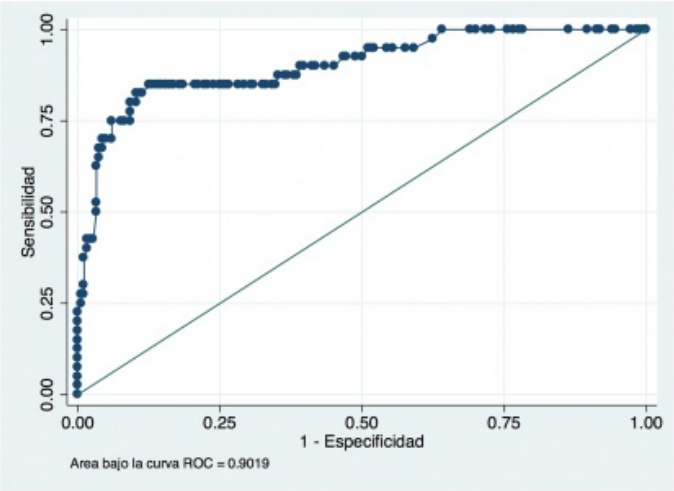
- **PPV**: Nos indica la proporción de verdaderos positivos del total que han sido clasificados como positivos, tanto verdaderos como falsos.

$$PPV = \frac{VP}{VP + FP}.$$

- **NPV**: Calcula de la proporción de verdaderos negativos del total que han sido clasificados como verdaderos y falsos negativos.

$$NPV = \frac{VN}{VN + FN}.$$

- **Área bajo la curva ROC**: También se conoce como AUC. En la representación de la curva ROC se observan todos los pares sensibilidad/especificidad obtenidos en los resultados observados. El eje x se situa $1 - especificidad$ y en el eje y la $sensibilidad$. De modo que cada punto de la gráfica representa el par $(1 - especificidad, sensibilidad)$, como podemos observar en la siguiente figura:



Cuanto más próxima esté la curva ROC a la esquina superior izquierda, más alta es la exactitud global del método, es decir, el área bajo la curva ROC debe ser lo más cercana a 1 posible.

- **Balance accuracy**: Es el valor medio de la sensibilidad y especificidad. Nos interesa que sea lo más cercano a 1, de esta forma tendremos certeza que el método es capaz de detectar a los verdaderos positivos y verdaderos negativos.

$$Balance \quad accuracy = \frac{Sensibilidad + Especificidad}{2}.$$

3.1. Implementación en R

Para realizar el estudio vamos a usar los siguientes recursos: R y Bioconductor.

Biocondutor es una colección de paquetes de R enfocado a la Bioinformática. Actualmente se desarrolla en uno de los institutos líderes en investigación sobre el cáncer, Fred Hutchinson Cancer Research Center, en Seatle.

Para comenzar con el análisis se necesita tener instalada algunas librerías que podemos encontrar en el repositorio de R y en Bioconductor. En el caso del R vamos a necesitar: `rpart, randomForest, caTools, caret, ROCR` y de Bioconductor: `DESeq2`. La manera en la que se instalan y se cargan es diferente en cada caso, como podemos ver:

```
#En R:
install.package("NombrePaquete")
library(NombrePaquete)

#En Bioconductor:
 source("http://bioconductor.org/biocLite.R")
 biocLite("NombrePaquete")
 library(NombrePaquete)
```

Una vez instaladas las librerías, procederemos a cargarlas.

```
library(DESeq2) ## Normalización
library(rpart) ## Ãrboles
library(randomForest) ## Random Forest y Bagging
library(caTools) ## Logitboost
library(ROCR) ## Para el área bajo la curva ROC
library(caret) ## Resto de tasas
```

Con el objetivo de facilitarnos posibles cambios en el futuro se definen los siguientes parámetros que nos serán de utilidad a lo largo del procedimiento, *n* es el número de individuos que están en el conjunto test y *kIter* es el número de iteraciones que se va a ejecutar cada método.

Con idea de mejorar la eficiencia de los resultados se realiza un número de iteraciones de los distintos métodos, haciendo un promedio global para estimar el resultado de los errores. Se va a hacer un bucle con tantas iteraciones como las que nos indica el parámetro kIter. En cada iteración se selecciona de forma aleatoria el conjunto de entrenamiento y el conjunto test.

Se han realizado pruebas con 100, 200 y 500 iteraciones, comprobando que a partir de 200 los resultados no se mejoran, por ello en el programa se ha tomado $kIter = 200$.

```
#Parámetros
n = 20 #Individuos en el conjunto test
kIter = 200 #Número de iteraaciones
```

A continuación, necesitamos descargar el paquete `MLSeq` que contiene el conjunto de datos en el siguiente enlace:

`https://www.bioconductor.org/packages/release/bioc/html/MLSeq.html`

La función `system.file` nos informa de la ruta donde se encuentra el archivo de texto *cervical.txt* que contiene los datos. Usamos el comando `read.table` para el que es necesario indicar el directorio que localiza los datos.

```
filepath = system.file("extdata/cervical.txt", package = "MLSeq")
cervical = read.table(filepath, header=TRUE)
```

Comenzamos con una breve estudio descriptivo del conjunto Cervical.

El conjunto de datos tienen 714 filas que se corresponden con los distintos genes $(let-7a, let-7a*, \ldots)$y 58 columnas que hace referencia a cada uno de los individuos de la muestra $(N1, \ldots, N29, T1, \ldots, T29)$.

```
dim(cervical)
```

```
[1] 714  58
```

Los datos son de la siguiente forma:

```
head(cervical)
```

```
         N1   N2   N3    N4   N5  N6 N7   N8   N9   N10  N11   N12   N13
let-7a  865  810 5505  6692 1456 588  9 4513 1962 10167 4113  2610  5008
let-7a*   3   12   30    73    6   2  0  199   10   173   30   105    71
let-7b  975 2790 4912 24286 1759 508 33 6162 1455 18110 8862 12481 21641
let-7b*  15   18   27   119   11   3  0  116   17   233   40   180   288
let-7c  828 1251 2973  6413  713 339 23 2002  476  3294 5929  1816  3278
let-7c*   0    0    0     1    0   0  0    3    0     3    0     0     3
```

Antes de continuar vamos a ver cual es la estructura de los datos. La función `summary` realiza un estudio descriptivo de cada una de las columnas. Se mostrará solo las primeras 8 columnas.

```
summary(cervical)
```

```
      N1                N2                 N3                N4          
 Min.   :   0.00   Min.   :    0.00   Min.   :    0.0   Min.   :    0.0  
 1st Qu.:   0.00   1st Qu.:    0.00   1st Qu.:    0.0   1st Qu.:    0.0  
 Median :   0.00   Median :    0.00   Median :    0.0   Median :    1.0  
 Mean   :  31.44   Mean   :   55.74   Mean   :  100.4   Mean   :  310.2  
 3rd Qu.:   2.00   3rd Qu.:    5.00   3rd Qu.:    7.0   3rd Qu.:   15.0  
 Max.   :6356.00   Max.   :11170.00   Max.   :11547.0   Max.   :51377.0  
       N5                N6                 N7                N8          
 Min.   :    0.00   Min.   :   0.00   Min.   :  0.000   Min.   :    0  
 1st Qu.:    0.00   1st Qu.:   0.00   1st Qu.:  0.000   1st Qu.:    0  
 Median :    1.00   Median :   0.00   Median :  0.000   Median :    2  
 Mean   :   74.06   Mean   :  12.81   Mean   :  1.852   Mean   :  175  
 3rd Qu.:    7.00   3rd Qu.:   1.00   3rd Qu.:  0.000   3rd Qu.:   16  
 Max.   :17214.00   Max.   :2433.00   Max.   :298.000   Max.   :22869  
```

```
class(cervical)
```

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

El siguiente paso será normalizar los datos brutos, es decir, estimar el valor o razón que lleva a una misma escala los conteos. De esta forma conseguimos hacer comparable los datos.

Para los datos de conteos tenemos distintos métodos de normalización, en este ejemplo se ha tomado el que aparece en la librería `DESeq2` como se comentó en el capítulo 1.

El proceso de normalizar necesita convertir a matriz el conjunto de datos usando `DESeqDataSetFromMatrix`. Dicho conjunto debe tener por filas los genes y por columnas cada muestra o individuo. De forma que, por ejemplo, para el valor de la i-ésima fila y j-ésima columna de la matriz de conteos nos dice las lecturas que tenemos para el gen i en la muestra j. Posteriormente se transforma a un objeto del tipo *deseq* a través de la función `DESeq`. Por último, la función `counts` nos devuelve la normalización.

```
class = data.frame(condition = factor(rep(c('N','T'), c(29,29))))
as.factor(class[,1]) #Se convierte en factor
cervicalS4 = DESeqDataSetFromMatrix(countData = cervical, colData = class,
                                    formula(~condition)) #Paso a matriz.
cervicalS4 = DESeq(cervicalS4, fitType = "local")
cervicalS4 #Datos transformados a la clase DESeq.
cervicalNorD = counts(cervicalS4, normalized = T) #Normalización.
cervical = data.frame(t(cervicalNorD)) #Paso a data.frame.
```

Se van a definir algunas matrices, donde se irán almacenando los valores que se han definido: sensibilidad, especificidad, área bajo la curva ROC, tasa global de aciertos, PPV, NPV, balance accuracy así, como la predicción de cada método para un número determinado de iteraciones.

```
#Matrices necesarias
sensibilidad = matrix(NA, nrow=kIter, ncol = 4)
colnames(sensibilidad) <- c("CART", "bagging", "RF", "Logitboost")

especificidad = matrix(NA, nrow=kIter, ncol = 4)
colnames(especificidad) <- c("CART", "bagging", "RF", "Logitboost")

auc = matrix(NA, nrow=kIter, ncol = 4)
colnames(auc) <- c("CART", "bagging", "RF", "Logitboost")

tasa_global_acierto = matrix(NA, nrow=kIter, ncol = 4)
colnames(tasa_global_acierto) <- c("CART", "bagging", "RF", "Logitboost")

PPV = matrix(NA, nrow=kIter, ncol = 4)
colnames(PPV) <- c("CART", "bagging", "RF", "Logitboost")

NPV = matrix(NA, nrow = kIter, ncol = 4)
colnames(NPV) <- c("CART", "bagging", "RF", "Logitboost")

balance_accuracy = matrix(NA, nrow = kIter, ncol = 4)
colnames(balance_accuracy) <- c("CART", "bagging", "RF", "Logitboost")

cart.pred_N = matrix(NA, nrow = n, ncol = kIter)

bagging.pred_N = matrix(NA, nrow = n, ncol = kIter)

rf.pred_N = matrix(NA, nrow = n, ncol = kIter)

boost.pred_N = matrix(NA, nrow = n, ncol = kIter)

cart.pred_T = matrix(NA, nrow = n, ncol = kIter)

bagging.pred_T = matrix(NA, nrow = n, ncol = kIter)

rf.pred_T = matrix(NA, nrow = n, ncol = kIter)

boost.pred_T = matrix(NA, nrow = n, ncol = kIter)
```

Necesitamos definir la variable dependiente o respuesta, que en este caso será si un individuo tiene tumor cervical

(T) o no (N) para posteriormente unirla al conjunto de datos *cervical*. Como en nuestro conjunto de datos aparecen ordenados, la variable respuesta viene dada por:

```
var.dep = factor(rep(c('N','T'), c(29,29)))
datos = data.frame(var.dep, cervical)
```

En cada iteración se ejecuta: Árboles de Clasificación, Bagging, Random Forest y Bossting. Los modelos se guardarán en las variables `tree.fit, bag.fit, rf.fit, boost.fit` respectivamente.

En el caso de Árboles de Clasificación, tal y como se ha visto en teoría se realiza la poda del árbol, `prun.fit` y este árbol podado será el que se use en la predicción de la clase de los individuos del conjunto test. En el resto de modelos se predice la muestra test con los modelos calculados directamente.

Posteriormente se medirá las tasas de aciertos. Para ello calculamos la sensibilidad, especificidad, área bajo la curva ROC, la tasa de acierto global, PPV, NPV y balance accuracy.

```
for (i in 1:kIter){

  cat("Iteración:", i, "\n")

  #Selección del conjunto de entrenamiento y test.
  set.seed(12345 + i)
  entren = sample(1:nrow(cervical), nrow(cervical)*2/3)
  m.ent = datos[entren,]
  m.test = datos[-entren,]
  v.dep.ent = var.dep[entren]
  v.dep.test = var.dep[-entren]

  ##### Árboles de clasificación:
  tree <- rpart(var.dep ~., m.ent, control = rpart.control(minsplit = 20,
                minbucket = 0.66, cp = 0.01, maxdepth = 30))
  prun.fit <- prune(tree, cp=tree$cptable[which.min(tree$cptable[,"xerror"]),
                                          "CP"])
  prun.pred = predict(prun.fit, m.test, type="class")

  #Tasas:
  confusion.matrix = confusionMatrix(prun.pred, v.dep.test, positive = 'T')
  sensibilidad[i,1] = confusion.matrix$byClass[1]
  especificidad[i,1] = confusion.matrix$byClass[2]
  tasa_global_acierto[i,1] = confusion.matrix$overall[1]
  PPV[i,1] = confusion.matrix$byClass[3]
  NPV[i,1] = confusion.matrix$byClass[4]
  balance_accuracy[i,1] = confusion.matrix$byClass[8]

  #AUC
  probarb = predict(prun.fit,m.test[-1], type="prob")
  cart.pred_N[,i] = probarb[,1]
  cart.pred_T[,i] = probarb[,2]
  pred <- prediction(probarb[,2], v.dep.test)
  perf <- performance(pred,'auc')
  auc[i,1] = as.numeric(perf@y.values)
```

```
#####Bagging:
bag.fit <- randomForest(var.dep ~., data = m.ent,
                        mtry = sqrt((dim(m.ent)[2])-1), tree = 500,
                        importance=TRUE, nodesize=1)
bag.pred = predict(bag.fit, m.test[-1])

#Tasas:
confusion.matrix = confusionMatrix(bag.pred, v.dep.test, positive = 'T')
sensibilidad[i,2] = confusion.matrix$byClass[1]
especificidad[i,2] = confusion.matrix$byClass[2]
tasa_global_acierto[i,2] = confusion.matrix$overall[1]
PPV[i,2] = confusion.matrix$byClass[3]
NPV[i,2] = confusion.matrix$byClass[4]
balance_accuracy[i,2] = confusion.matrix$byClass[8]

#AUC
probbag <- predict(bag.fit, m.test[-1], type="prob")
bagging.pred_N[,i] = probbag[,1]
bagging.pred_T[,i] = probbag[,2]
pred <- prediction(probbag[,2], v.dep.test)
perf <- performance(pred,'auc')
auc[i,2] = as.numeric(perf@y.values)


#####Random Forest:
randfor.fit = randomForest(var.dep ~., data = m.ent,
                           mtry = sqrt((dim(m.ent)[2])-1),
                           importance=TRUE, ntree = 500)
randfor.pred = predict(randfor.fit, m.test)

#Tasas:
confusion.matrix = confusionMatrix(randfor.pred, v.dep.test,
                                   positive = 'T')
sensibilidad[i,3] = confusion.matrix$byClass[1]
especificidad[i,3] = confusion.matrix$byClass[2]
tasa_global_acierto[i,3] = confusion.matrix$overall[1]
PPV[i,3] = confusion.matrix$byClass[3]
NPV[i,3] = confusion.matrix$byClass[4]
balance_accuracy[i,3] = confusion.matrix$byClass[8]

#AUC:
probran = predict(randfor.fit, m.test[-1], type="prob")
rf.pred_N[,i] = probran[,1]
rf.pred_T[,i] = probran[,2]
pred = prediction(probran[,2], v.dep.test)
perf = performance(pred,'auc')
auc[i,3] = as.numeric(perf@y.values[[1]])


#####Logitboost:
```

```
  boost.fit = LogitBoost(m.ent[,-1], m.ent[,1], nIter = ncol(m.ent))
  boost.pred = predict(boost.fit, m.test[,-1])

  #Tasas:
  confusion.matrix = confusionMatrix(boost.pred, v.dep.test,
                                     positive = 'T')
  sensibilidad[i,4] = confusion.matrix$byClass[1]
  especificidad[i,4] = confusion.matrix$byClass[2]
  tasa_global_acierto[i,4] = confusion.matrix$overall[1]
  PPV[i,4] = confusion.matrix$byClass[3]
  NPV[i,4] = confusion.matrix$byClass[4]
  balance_accuracy[i,4] = confusion.matrix$byClass[8]

  #AUC:
  proboost = predict(boost.fit, m.test[-1], type = 'raw')
  boost.pred_N[,i] = proboost[,1]
  boost.pred_T[,i] = proboost[,2]
  pred = prediction(proboost[,2], v.dep.test)
  perf = performance(pred,'auc')
  auc[i,4] = as.numeric(perf@y.values[[1]])
}
```

Una vez ejecutados los métodos realizamos la media de las probabilidades de la pertenencia a cada una de las clases de cada uno de los métodos, de esta forma conseguimos unificar la predicción para cada clase.

```
#Media de las prediciones por clase :
cart.pred_N = rowMeans(cart.pred_N, dim(cart.pred_N)[1])
cart.pred_T = rowMeans(cart.pred_T, dim(cart.pred_T)[1])

bagging.pred_N = rowMeans(bagging.pred_N, dim(bagging.pred_N)[1])
bagging.pred_T = rowMeans(bagging.pred_T, dim(bagging.pred_T)[1])

rf.pred_N = rowMeans(rf.pred_N, dim(rf.pred_N)[1])
rf.pred_T = rowMeans(rf.pred_T, dim(rf.pred_T)[1])

boost.pred_N = rowMeans(boost.pred_N, dim(boost.pred_N)[1])
boost.pred_T = rowMeans(boost.pred_T, dim(boost.pred_T)[1])
```

Mostramos por columnas los resultados anteriormente citados para las dos clases en los distintos métodos.

```
cart.pred_final = cbind(cart.pred_N, cart.pred_T)
```

```
      cart.pred_N cart.pred_T
 [1,]   0.8443354   0.1556646
 [2,]   0.8433643   0.1566357
 [3,]   0.8016736   0.1983264
 [4,]   0.7591872   0.2408128
 [5,]   0.8157354   0.1842646
 [6,]   0.8203614   0.1796386
 [7,]   0.8339871   0.1660129
 [8,]   0.7910445   0.2089555
```

```
 [9,]   0.6650945   0.3349055
[10,]   0.5651860   0.4348140
[11,]   0.3925030   0.6074970
[12,]   0.3180283   0.6819717
[13,]   0.3012208   0.6987792
[14,]   0.2648579   0.7351421
[15,]   0.2725495   0.7274505
[16,]   0.2488240   0.7511760
[17,]   0.2249578   0.7750422
[18,]   0.2077848   0.7922152
[19,]   0.2047326   0.7952674
[20,]   0.2461546   0.7538454
```

```
bagging.pred_final = cbind(bagging.pred_N, bagging.pred_T)
```

```
 bagging.pred_N bagging.pred_T
 [1,]        0.80562        0.19438
 [2,]        0.78849        0.21151
 [3,]        0.75788        0.24212
 [4,]        0.72548        0.27452
 [5,]        0.78799        0.21201
 [6,]        0.79194        0.20806
 [7,]        0.78253        0.21747
 [8,]        0.74571        0.25429
 [9,]        0.67294        0.32706
[10,]        0.55539        0.44461
[11,]        0.41807        0.58193
[12,]        0.36296        0.63704
[13,]        0.32982        0.67018
[14,]        0.30189        0.69811
[15,]        0.27048        0.72952
[16,]        0.24553        0.75447
[17,]        0.23063        0.76937
[18,]        0.25465        0.74535
[19,]        0.21242        0.78758
[20,]        0.26517        0.73483
```

```
rf.pred_final = cbind(rf.pred_N, rf.pred_T)
```

```
      rf.pred_N rf.pred_T
 [1,]   0.80591   0.19409
 [2,]   0.78946   0.21054
 [3,]   0.75909   0.24091
 [4,]   0.72395   0.27605
 [5,]   0.78879   0.21121
 [6,]   0.79290   0.20710
 [7,]   0.78322   0.21678
 [8,]   0.74977   0.25023
 [9,]   0.67740   0.32260
```

```
[10,]   0.55728   0.44272
[11,]   0.41697   0.58303
[12,]   0.36196   0.63804
[13,]   0.32998   0.67002
[14,]   0.30700   0.69300
[15,]   0.27006   0.72994
[16,]   0.24279   0.75721
[17,]   0.22817   0.77183
[18,]   0.25493   0.74507
[19,]   0.21440   0.78560
[20,]   0.26809   0.73191
```

```
boost.pred_final = cbind(boost.pred_N, boost.pred_T)
```

```
> boost.pred_final
       boost.pred_N boost.pred_T
 [1,]    0.9899659   0.01003408
 [2,]    0.9453041   0.05469594
 [3,]    0.9449994   0.05500062
 [4,]    0.8681382   0.13186178
 [5,]    0.9239214   0.07607859
 [6,]    0.9454028   0.05459723
 [7,]    0.9413447   0.05865528
 [8,]    0.9286750   0.07132499
 [9,]    0.8263493   0.17365074
[10,]    0.6463167   0.35368333
[11,]    0.4746514   0.52534863
[12,]    0.3550151   0.64498488
[13,]    0.2972174   0.70278260
[14,]    0.3154730   0.68452704
[15,]    0.2957635   0.70423653
[16,]    0.2365556   0.76344437
[17,]    0.2425209   0.75747915
[18,]    0.2002934   0.79970660
[19,]    0.1325892   0.86741084
[20,]    0.2234594   0.77654056
```

Las tasas que describimos al principio de la sección se han ido calculando en cada iteración y para cada uno de los métodos. A continuación se realizan las medias de las tasas mediante la información almacenada en las distintas iteraciones. Utilizamos como variable de almacenamiento el nombre de la tasa acabado en m.

```
#Tasas:

#Media de las matrices:
sensibilidadm = colMeans(sensibilidad, dim(sensibilidad)[2])
```

```
> sensibilidadm
      CART    bagging         RF LogitBoost
 0.7642916  0.9212784  0.9188393  0.7799762
```

```
especificidadm = colMeans(especificidad, dim(especificidad)[2])
```

```
> especificidadm
      CART    bagging         RF LogitBoost
 0.8421583  0.9442401  0.9477012  0.9539862
```

```
aucm = colMeans(auc, dim(auc)[2])
```

```
> aucm
      CART    bagging         RF LogitBoost
 0.8019884  0.9888224  0.9897835  0.9187912
```

```
tasa_global_aciertom = colMeans(tasa_global_acierto, dim(tasa_global_acierto)[2])
```

```
> tasa_global_aciertom
      CART    bagging         RF LogitBoost
   0.79900    0.92850    0.92925    0.86525
```

```
PPVm = colMeans(PPV, dim(PPV)[2])
```

```
> PPVm
      CART    bagging         RF LogitBoost
 0.8354225  0.9401714  0.9440492  0.9478945
```

```
NPVm = colMeans(NPV, dim(NPV)[2])
```

```
> NPVm
      CART    bagging         RF LogitBoost
 0.7842753  0.9220601  0.9200678  0.8125124
```

```
balance_accuracym = colMeans(balance_accuracy, dim(balance_accuracy)[2])
```

```
> balance_accuracym
      CART    bagging         RF LogitBoost
 0.8032250  0.9327592  0.9332702  0.8669812
```

3.2. Análisis de los resultados

En esta sección se van a analizar los resultados obtenidos, para ello se va a usar varios recursos. En primer lugar se verá una tabla de comparación, diagramas de barras y diagramas de cajas y bigotes.

Tabla con las tasas calculadas para todos los métodos.

	CART	Bagging	Random F.	Boosting
Sensibilidad	0.7693	0.9212	0.9188	0.7799
Especificidad	0.8421	0.9442	0.9477	0.9539
AUC	0.8019	0.9888	0.9897	0.9187
Tasa global de aciertos	0.7990	0.9285	0.9292	0.8652
PPV	0.8354	0.9401	0.9440	0.9478
NPV	0.7842	0.9220	0.9200	0.8125
Balance Accuracy	0.8032	0.9327	0.9332	0.8669

Se observan que todos los métodos funcionan muy bien en la clasificación con este conjunto de datos, destacamos Random Forest y Bagging que es donde encontramos las tasas más altas, es decir, los valores más próximos a 1.

Representaciones gráficas

En esta sección vamos a mostrar un diagrama de barras para cada una de las tasas calculadas en los diferentes métodos de clasificación.

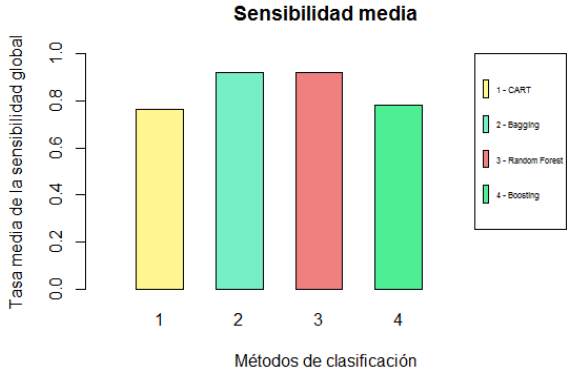


Figura 3.1: Sensibilidad media para los métodos.

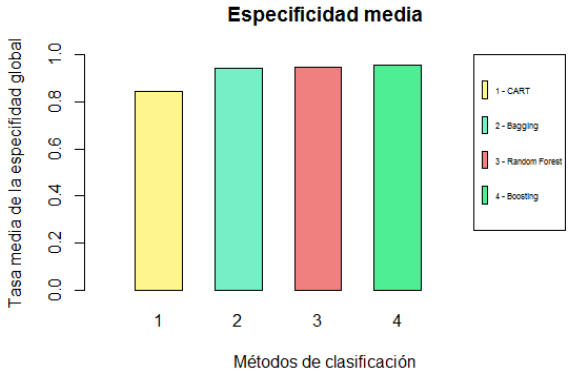


Figura 3.2: Especificidad media para los métodos.

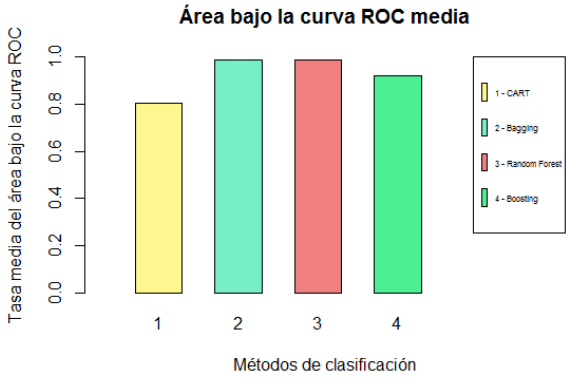


Figura 3.3: Área bajo la curva ROC media para los métodos.

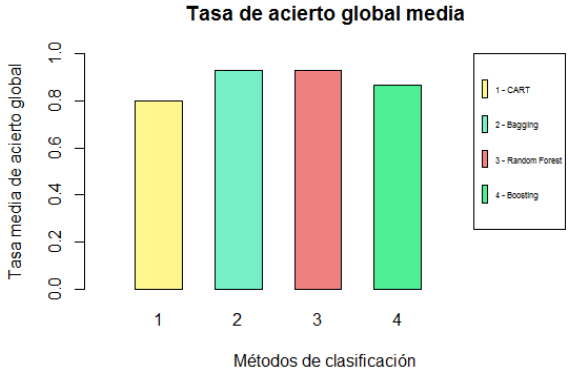


Figura 3.4: Media del Acierto Global para los métodos.

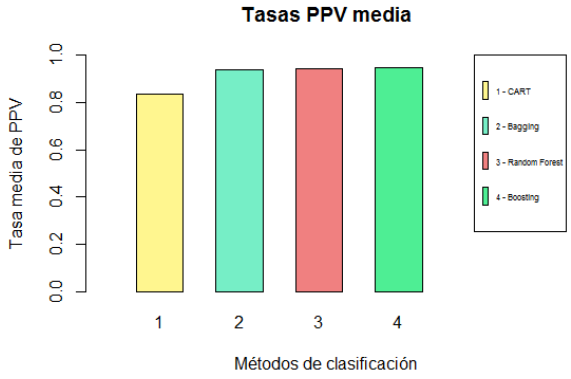


Figura 3.5: Media de PPV para los métodos.

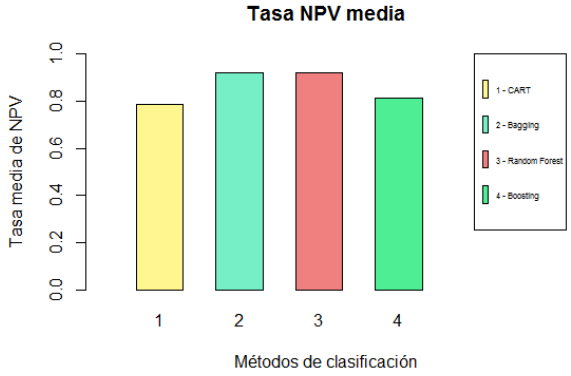


Figura 3.6: Media de NPV para los métodos.

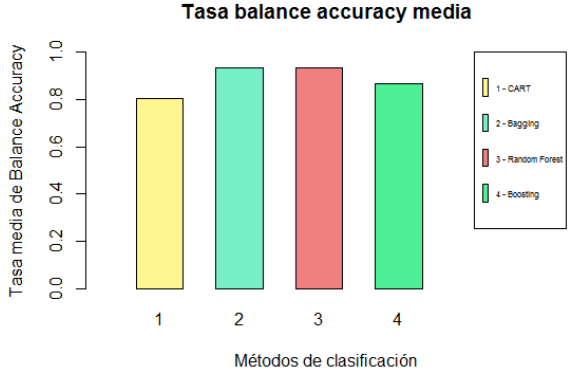


Figura 3.7: Balance Accuracy media para los métodos.

En estas gráficas se pueden observar las diferentes tasas para cada uno de los métodos. En general, se aprecia que para la mayoría de las tasas los mejores modelos son Bagging y Random Forest, siendo éste último sutilmente mejor. Para estos modelos el valor de las tasas está muy próximo a 1. Para el resto de los modelos ls resultados obtenidos son bastante buenos también.

Representación de la tasa media de acierto global frente al área media bajo la curva ROC

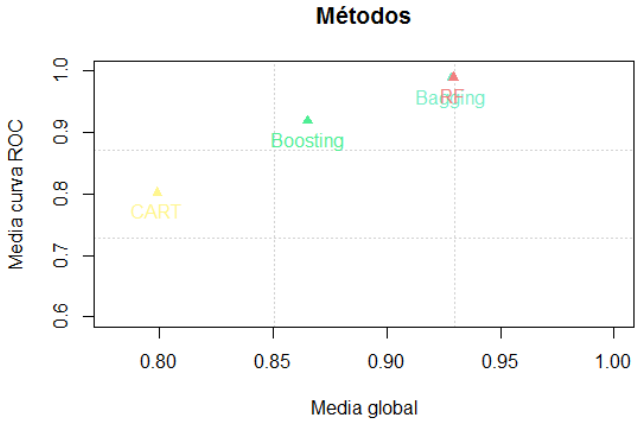


Figura 3.8:

Se ve que para la relación entre el área bajo la curva ROC y la tasa global de aciertos los métodos con tasas más altas son Bagging y Random Forest, mientras que el que tiene los valores más bajos es el método basado en Árboles.

Representación de la tasa media de Sensibilidad frente a la tasa media de Especificidad

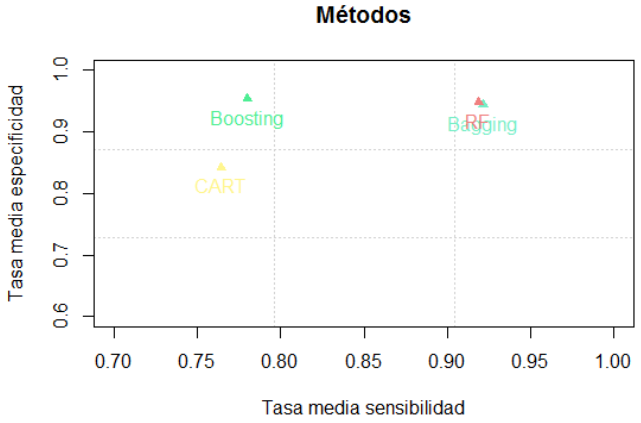


Figura 3.9:

Se aprecia que el método de Bagging y Random Forest presentan los valores más altos tanto en la especificidad como en la sensibilidad, luego para detectar los verdaderos positivos y negativos es útil usar alguno de estos métodos. El método de Boosting tiene un valor alto en la especificidad pero no tan alto en la sensibilidad. Por último el método basado en Árboles es el menos eficiente.

Diagramas de cajas

A continuación vamos a representar el resultado de las 200 iteraciones mediante diagramas de cajas y bigotes para cada una de las tasas en los diferentes métodos.

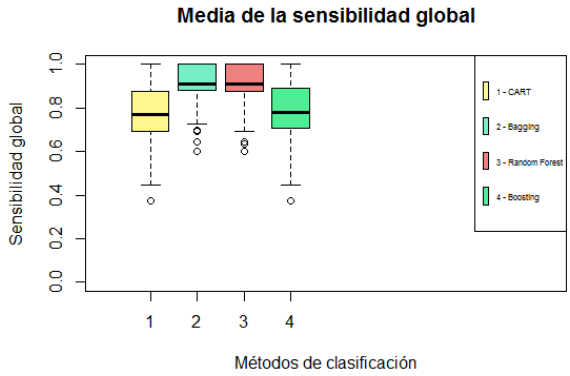


Figura 3.10: Tasa de Sensibilidad.

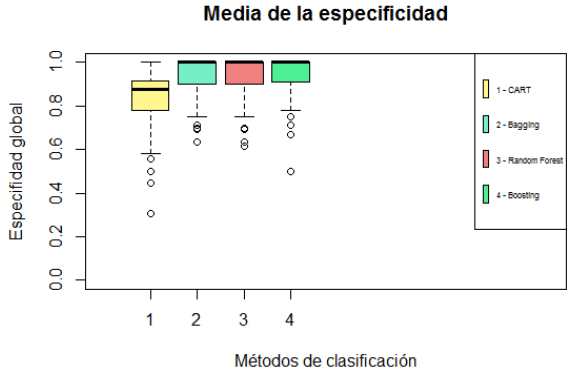


Figura 3.11: Tasa de Especificidad.

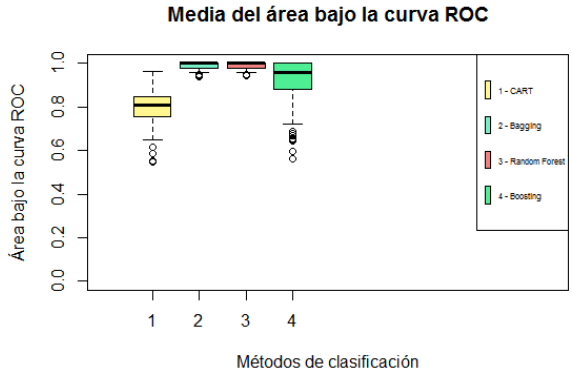


Figura 3.12: Área bajo la curva ROC.

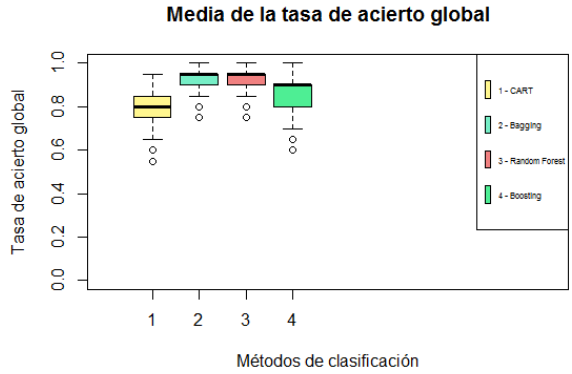


Figura 3.13: Tasa Global de Acierto.

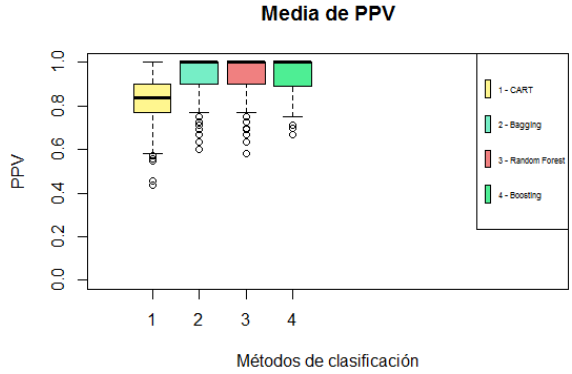


Figura 3.14: Tasa de PPV.

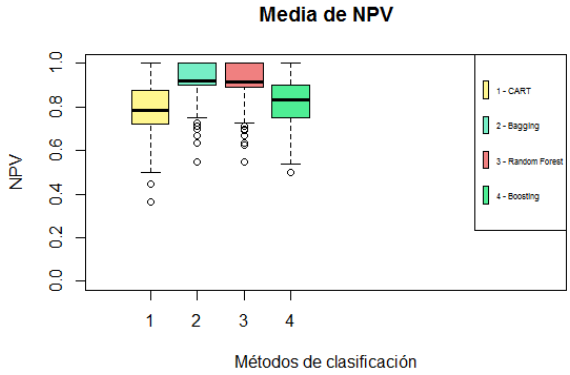


Figura 3.15: Tasa de NPV.

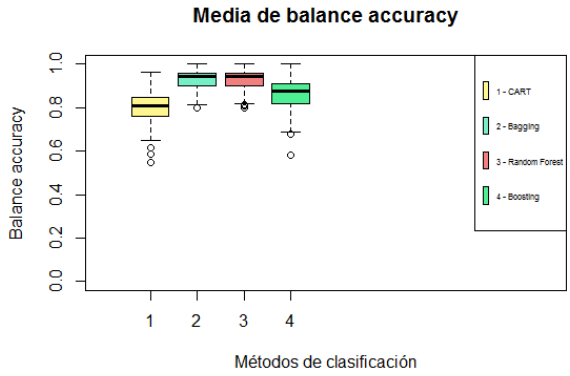


Figura 3.16: Balance Accuracy.

En general, se puede apreciar que para los métodos de Bagging y Random Forest los valores de la mayoría de las tasas están más concentrados. Mientras que para CART y Boosting los valores se encuentran más dispersos. En todas las gráficas se pueden observar valores atípicos para todos los métodos. Los valores más altos se vuelven a alcanzar con los métodos de Bagging y Random Forest como ya se ha visto anteriormente en las otras gráficas.

Capítulo 4

Paquetes de R

En esta sección se hace un listado de los paquetes de R que hemos usado en la memoria, explicando en qué consiste cada uno y detallando las funciones y sus parámetros asociados.

Paquete " caret "

Incluye funciones para ejecutar modelos de clasificación y regresión y representación de gráficos.

Funciones:

`confusion.matrix`: Calcula la matriz de confusión de las clases observadas y predichas con las estadísticas asociadas.

1. Uso: `confusionMatrix(data, reference)`
2. Argumentos:
    - `data` : un factor de clases predichas del método por defecto.
    - `reference` : un factor de clases que será usado como los resultados verdaderos.
3. Parámetros de salida: tasas de sensibilidad, especificidad, valor predictivo positivo y negativo, detection rate, detection prevelance, prevelancia, balance accuracy.

Paquete " caTools "

En este paquete hay una gran variedad de funciones implementadas, entre ellas podemos encontrar el algoritmo Friedman, Logitboost.

Funciones:

`LogitBoost`: Ajusta el modelo de Boosting mediante el algoritmo Logitboost para árboles de clasificación.

1. Uso: `LogitBoost(xlearn, ylearn, nIter=ncol(xlearn))`
2. Argumentos:
    - `xlearn` : una matriz o un data.frame que contiene al conjunto de datos.

- `ylearn` : un vector que indique la variable respuesta.
- `nIter` : un valor entero, que nos indique el número de veces que el algoritmo debe ser ejecutado.

`predict`: Clasifica un data frame usando un objeto del tipo LogitBoost que haya sido ajustado previamente.

1. Uso: `predict(object, xtest)`
2. Argumentos:
   - `object` : modelo con el que queramos hacer la clasificación, debe ser del tipo LogitBoost.
   - `xtest` : data frame que contiene los individuos a los que le queremos hacer la clasificación.

Paquete " Deseq2 "

Este paquete está incluido en Bioconductor. Se usa para analizar datos de conteo obtenidos en ensayos de secuenciación de alto rendimiento incluyendo pruebas de análisis de expresión diferencial. Se puede obtener más información de este paquete en el siguiente enlace:

`https://bioconductor.org/packages/release/bioc/html/DESeq2.html`

Funciones:

`counts`: Almacena los datos de conteo en una matriz de números enteros no negativos, una fila para cada observación y una columna para cada muestra.

1. Uso: `counts(object, normalized)`
2. Argumentos:
   - `object` : un objeto DESeqDataSet.
   - `normalized` :valor lógico que indica si queremos dividir los conteos por el tamaño de los factores o normalizarlos antes de devolverlos.

`Deseq`: Se usa para el estudio de genes diferencialmente expresados. En este trabajo se utilizará exclusivamente para obtener los datos normalizados.

1. Uso: `DESeq(object, fitType = c("parametric", "local", "mean")`
2. Argumentos:
   - `object` : un objeto de clase DESeqDataSet.
   - `fitType` : puede tomar varios valores, "parametric", "local.º "mean"para el ajuste de dispersión de intensidad de la media.

`DeseqDataSetFromMatrix`: Crea un objeto de la clase *DESeqDataSet* a partir de la matriz de conteos.

1. Uso: `DESeqDataSetFromMatrix(countData, colData, design)`
2. Argumentos:
   - `countData` : matriz de entrada, valores enteros no negativos.

- `colData` : DataFrame o data frame con al menos una columna. Las filas de colData corresponden a las columnas de countData.
- `design` : una fórmula que expresa cómo los recuentos para cada gen dependen de las variables de colData.

Paquete " randomForest "

Este paquete permite realizar clasificación y regresión mediante técnicas basadas en árboles, incluye Bagging y Random Forest.

Funciones:

`randomForest`: Se utliza tanto para ajustar un modelo de Bagging como de Random Forest.

1. Uso: `randomForest(x, data, mtry, ntree, importance, nodesize)`
2. Argumentos:
   - `data` : data frame que contiene las variables del modelo.
   - `mtry` : muestra de variables seleccionadas aleatoriamente como candidatas en cada división.
   - `ntree` : número de árboles.
   - `importance` : Valor lógico que indica si quiere almacenar la importancia de los predictores.
   - `nodesize` : Mínimo número de individuos en el nodo final.

`predict`: Predicción de los datos test usando random forest.

1. Uso: `predict(object, newdata)`
2. Argumentos:
   - `object` : modelo de la clase randomForest.
   - `newdata` : data frame que contiene los valores con los que se quieren obtener unas predicciones.
   - `type` : cadena de caracteres que indica el tipo de valor que quiere que sea devuelto (¿¿o depende del problema que se está tratando??). Puede ser " vector ", " prob ", " class ", " matrix ".

Paquete " ROCR "

En este paquete encontramos herramientas para realizar gráficos de la curva ROC y para la sensibilidad, especificidad de las curvas, entre otras opciones.

Funciones:

`performance`: Todos los tipos de evaluaciones de predicción se realizan con esta función.

1. Uso: `performance(prediction.obj, measure)`
2. Argumentos:

- `prediction.obj` : un objeto de clase prediction.
- `measure` : medida con la que queremos evaluar la predicción.

`prediction`: Esta función se usa para transformar los datos de entrada (que pueden ser vectores, matrices, data frames o en forma de lista) en un formato estandarizado.

1. Uso: `prediction(predictions, labels)`
2. Argumentos:
   - `predictions` : un vector, matriz, lista o data frame que contiene las predicciones.
   - `labels` : un vector, matriz, lista o data frame que contiene la clasificación verdadera.

Paquete " rpart "

Incluye métodos de división recursiva para clasificación, regresión y "árboles de supervivencia". En este paquete encontramos la mayoría de las funciones descritas en el libro de Breiman, Friedman, Olshen y Stone en 1984.

Funciones:

`rpart`: Ajusta un modelo basado en árboles de clasificación.

1. Uso: `rpart(formula, data, control = c(minsplit, minbucket, cp, maxdepth)`
2. Argumentos:
   - `formula` : una fórmula donde se expresa la variable dependiente o respuesta en función de las variables independientes.
   - `data` : data frame donde se encuentran las variables almacenadas.
   - `control` : lista de opciones que controlan detalles del algoritmo.
   - `minsplit` : mínimo número de observaciones que deben existir en un nodo para la que la división sea efectuada.
   - `minbucket` : mínimo número de observaciones que hay en un nodo final.
   - `cp` : complejidad del árbol.
   - `maxdepth` : profundidad máxima del árbol.

`prune`: Determina una secuencia anidada de subárboles, cortando de forma recursiva las divisiones menos importantes, basándose en el parámetro de complejidad (cp).

1. Uso: `prune(tree, cp)`
2. Argumentos:
   - `tree` : un modelo de la clase rpart.
   - `cp` : parámetro de complejidad con el que se va a cortar el árbol.

`predict`: Devuelve un vector con las respuestas estimadas a partir de un objeto de rpart.

1. Uso: `predict(object, newdata, type)`
2. Argumentos:
   - `object` : modelo de la clase rpart.
   - `newdata` : data frame que contiene los valores con los que se quieren obtener unas predicciones.
   - `type` : cadena de caracteres que indica el tipo de valor que quiere que sea devuelto. Puede ser " vector ", " prob ", " class ", " matrix ".

Bibliografía

[1] Breiman, L., (1994) *Bagging predictors*. Recuperado de: http://statistics.berkeley.edu/sites/default/files/tech-reports/421.pdf

[2] Breiman, L., (2001). *Random Forests*. Recuperado de: https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

[3] Breiman, L., Friedman. J., Stone, C.S., y Olshen, R.A. (1998). *Classification and Regression Trees*. Belmont, Wadsworth International Group.

[4] Dettling, M., y Bühlmann, P., (2002). *Boosting for tumor classification with gene expression data*. Seminar für Statistik, ETH Zürich, Switzerland.

[5] Friedman, J., Hastie, T., y Tibshirani, R., (2000). *Additive logistic regression: A statistical view of boosting*. Annals of Statistics, (pp.337–374).

[6] Hastie, T., Tibshirani, R., y Friedman, J., (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics.

[7] Illumina. *An Introduction to Next-Generation Sequencing Technology*. Recuperado de: http://www.illumina.com/content/dam/illumina-marketing/documents/products/illumina_sequencing_introduction.pdf

[8] Instituto de Biotecnología-UNAM. *Métodos fisicoquímicos en biotecnología: Secuenciación de Ácidos Nucleicos*. Recuperado de: http://www.ibt.unam.mx/computo/pdfs/met/secuenciacion_acidos_nucleicos.pdf

[9] New England Biolabs. *Getting Started with RNA-Seq*. Recuperado de: https://www.neb.com/tools-and-resources/usage-guidelines/getting-started-with-rna-seq

[10] R-Project. *Available CRAN Packages By Name*. Recuperado de: https://cran.r-project.org/

[11] Therneau, T.M., Atkinson, E.J., y Mayo Foundation *An Introduction to Recursive Partitioning Using the RPART Routines*,https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf

[12] Velasco, J.M., Romero, T., Salamanca, C., y López, R., (2009). *Biología 2º Bachillerato*. Editorial Editex.

[13] Zararsiz, G., Goksuluk, D., Korkmaz, S., Eldem, V., Duru, I.P., Unver, T., y Ozturk, A., (2016). *Machine learning interface for RNA-Seq data*. Recuperado de: https://www.bioconductor.org/packages/release/bioc/html/MLSeq.html