FACULTAD DE MATEMÁTICAS

DOBLE GRADO DE MATEMÁTICAS Y ESTADISTÍCA

DEPARTAMENTO DE ESTADISTICA E
INVESTIGACIÓN OPERATIVA

TÉCNICAS BOOSTING

TRABAJO FIN DE GRADO

Manuel Pérez García

Tutor: Rafael Pino Mejías

Sevilla, Diciembre 2018

Contenido

1. RESUMEN

En este documento se tratará en un principio, de manera general, las principales características de los métodos basados en la combinación de modelos (Bagging, Random Forest y Boosting).

Básicamente, los tres métodos se sustentan en la combinación de árboles de decisión. Los árboles de decisión son una manera muy simple y práctica de realizar predicciones, aunque cierto es que los resultados son muy mejorables. Es por ello por lo que surgen este tipo de métodos (combinación de modelos), para intentar mejorar los resultados de las predicciones.

Los árboles de decisión, en función de la variable a predecir (variable objetivo), se pueden dividir en árboles de regresión y árboles de clasificación. En este documento se expondrán los procesos de creación de cada uno de ellos, como también se detallará la forma de predecir las distintas observaciones de las que dispongamos.

Tras explicar los modelos (árboles) con los que suelen trabajar los métodos de combinación de modelos, se expondrán, de manera muy generalizada, los modelos de Bagging y Random Forest. Veremos sus principales similitudes y diferencias tanto entre ellos como con el Boosting, el cual es el principal componente de este documento.

En cuanto al Boosting, se explicarán de manera muy detallada todas sus características, su procedencia y sus distintos tipos de técnicas y algoritmos, los cuales se expondrán uno a uno haciendo hincapié en sus diferencias y similitudes.

Por último, se hará uso del software R para realizar dos ejemplos prácticos. Uno de ellos estará basado en la clasificación de incidencias de una de las mayores empresas de telecomunicaciones de España. Además, en él se llevará a cabo un atractivo proceso de preprocesamiento y limpieza de los datos. El otro ejemplo estará basado en el conjunto de datos “Carseats” contenido en la librería ISLR de R, el cual se basará en la clasificación de una variable binaria.

2. SUMMARY

In this document, the main characteristics of the ensemble models (Bagging, Random Forest and Boosting) will be discussed in a general way.

Basically, all the three methods are based on the combination of decision trees. Decision trees are a very simple and practical way to make predictions, although it is true that the results are very improvable. That is why this type of methods (ensemble models) arise. They try to improve the results of the predictions of a single decisión tree.

Decision trees, depending on the variable to be predicted (target variable), can be divided into regression trees and classification trees. In this document the processes of creation of each of them will be exposed, as well as the way of predicting the different observations that we have.

After explaining the models (trees) with which the ensemble models usually work, the models of Bagging and Random Forest will be exposed in a very general way. We will see their main similarities and differences both between them and with Boosting, which is the main component of this document.

As for Boosting, all its characteristics, its origin and its different types of techniques and algorithms will be explained in a very detailed manner, which will be presented one by one, emphasizing their differences and similarities.

Finally, the R software will be used to show two practical examples. One of them will be based on the classification of incidents of one of the largest telecommunications companies in Spain. In addition, an attractive process of preprocessing and data cleaning will be carried out on it. The other examples will be based on the "Carseats" data set contained in ISLR package of R, which will be based on the classification of a binary variable.

3. INTRODUCCIÓN

Las técnicas de combinación de modelos consisten en la agregación de un cierto número de modelos para obtener una clasificación o predicción a partir de los distintos clasificadores o predictores previamente generados.

La construcción de cada uno de los modelos elementales puede depender de aspectos como los siguientes:

- Definición del conjunto de entrenamiento (muestras bootstrap, reponderación)
- Selección de variables
- Definición del conjunto test
- Elección del modelo (por ejemplo, todos árboles de decisión, o una mezcla de modelos de distinta naturaleza)

El principal objetivo de este tipo de técnicas es disminuir el error de generalización. A continuación, se describen algunas ideas en el contexto de predicción de variables cuantitativas (regresión).

Dado un conjunto de entrenamiento $D = \{D_i = (X_i, Y_i),\ i = 1,2, \dots, n\}$ con X p-dimensional e Y real (problema de regresión) se consideran M modelos de predicción $g_1(x)$,…, $g_M(x)$ para $E(Y|X = x)$. A partir de los *M* modelos se trata de construir un predictor mediante algún tipo de combinación.

Por ejemplo, tanto la función de densidad como el valor esperado de *Y* se pueden expresar en función de las probabilidades de cada modelo:

$$f(Y|x, D) = \sum_{m=1}^{M} f(Y|x, g_m, D)\ P(g_m|D)$$

$$E(Y|x, D) = \sum_{m=1}^{M} E(Y|x, g_m, D)\ P(g_m|D)$$

Esta última expresión nos conduce a que una estimación del valor esperado de Y sería:

$$\hat{E}(Y|x, D) = \sum_{m=1}^{M} \hat{E}(Y|x, g_m, D)\ \hat{P}(g_m|D)$$

En cuanto al término $\hat{E}(Y|x, g_m, D)$, cada modelo proporciona directamente una estimación de cada valor esperado. Por tanto,

$$\hat{E}(Y|x, g_m, D) = g_m(x)$$

Ahora bien, ¿cómo podemos estimar la probabilidad de cada modelo?, En definitiva, nos estaríamos refiriendo al peso de cada uno. En principio, le podríamos dar el mismo peso a cada uno de ellos, es decir,

$$\hat{P}(g_m|D) = \frac{1}{M}, m = 1, \dots, M$$

Sin embargo, es poco apropiado debido a que podemos tener modelos de distinta complejidad. Una alternativa a lo anterior sería considerar el criterio BIC (Criterio de Información Bayesiano), que depende tanto de la calidad del ajuste como de la complejidad:

$$\hat{P}(g_m|D) = \frac{e^{-2BIC_m}}{\sum_{l=1}^{M} e^{-2BIC_l}}$$

Otra opción consiste en ajustar un modelo de regresión a partir de los M predictores. Por ejemplo, mediante regresión lineal múltiple, se trata de identificar la combinación lineal de menor ECM:

$$\hat{w} = \arg min_w \, E\left(Y - \sum_{m=1}^{M} w_m g_m(x)\right)^2$$

$$= E\left[\begin{pmatrix} g_1(x) \\ \vdots \\ g_M(x) \end{pmatrix}(g_1(x) \quad \dots \quad g_M(x))\right]^{-1} E\left[\begin{pmatrix} g_1(x) \\ \vdots \\ g_M(x) \end{pmatrix} Y\right]$$

Este modelo, por definición, tiene menor error esperado que cualquiera de los simples:

$$E\left(Y - \sum_{m=1}^{M} \hat{w}_m g_m(x)\right)^2 \leq E\big(Y - g_m(x)\big)^2 \; \forall m = 1, \dots, M$$

Dado que el modelo poblacional no se podrá calcular, parece natural aproximarla por el ajuste sobre los datos disponibles:

$$\widehat{w} = \arg min_w \sum_{i=1}^{n} \left(y_i - \sum_{m=1}^{M} w_m g_m(x_i) \right)^2$$

Sin embargo, esta aproximación falla en ciertas situaciones. Por ejemplo, si cada modelo es el que corresponde a la mejor predicción con m variables de entre M variables en total. En tal caso todos los coeficientes estimados w_m serán 0 excepto el del modelo M. Para intentar solventar este problema, que viene dado por la distinta complejidad, se puede utilizar lo que se llama apilamiento (*stacking*):

$$\widehat{w}^{st} = \arg min_w \sum_{i=1}^{n} \left(y_i - \sum_{m=1}^{M} w_m g_m^{-i}(x_i) \right)^2$$

g_m^{-i} denota el m-ésimo modelo construído sobre el conjunto de datos sin la observación i-ésima. Las predicciones finales serían de la forma:

$$\sum_{m=1}^{M} \widehat{w}_m^{st} g_m(x)$$

Esta técnica evita dar demasiado peso a los modelos más complejos. Está muy relacionada con n-validación cruzada (Jackknife): si los vectores de coeficientes se restringen a que $M - 1$ sean nulos y otro igual a 1, conduce a una selección de modelos por menor error mediante n -validación cruzada.

Si los coeficientes se restringen a formar una distribución de probabilidad, los coeficientes proporcionan una estimación de la probabilidad a posteriori de cada modelo (resoluble mediante programación cuadrática).

En vez de regresión lineal, también se podría utilizar otro tipo de técnica.

A lo largo de este documento nos centraremos en explicar el Boosting, que se trata de una técnica de este tipo. Además, lo compararemos con otros métodos similares como pueden ser el Bagging o el Random Forest, donde se construyen múltiples modelos del mismo tipo usando diferentes submuestras (bootstrap) del conjunto de entrenamiento.

4. ÁRBOLES

Como se ha presentado en la sección anterior, las técnicas de combinación de modelos consisten en la agregación de un cierto número de modelos para obtener una clasificación o predicción a partir de los distintos clasificadores o predictores previamente generados.

A lo largo de este documento, los modelos que vamos a utilizar van a ser los árboles, ya que, por sí solos no tienen una alta capacidad predictiva pero su combinación puede llegar a ser muy fiable.

Los métodos basados en árboles para la regresión y la clasificación implican estratificar o segmentar el espacio que forman las variables predictoras en varias regiones simples. Para hacer una predicción de una observación dada, normalmente usamos la media (regresión) o la moda (clasificación) de las observaciones pertenecientes al conjunto de entrenamiento en la región a la que pertenece tal observación.

Estos métodos son simples y útiles para la interpretación. Sin embargo, como hemos mencionado al principio de la sección, generalmente no son competitivos con los mejores modelos de aprendizaje supervisado. Entre las diversas metodologías existentes, se describe a continuación la metodología CART (Breiman et al., 1984).

Árboles de Regresión

El proceso de construcción de los árboles de regresión persigue:

1- Dividir el espacio de las variables predictoras $X_1, \dots, X_p$ en J regiones disjuntas $R_1, \dots, R_J$.
2- Todas las observaciones que caigan en la misma región tendrán el mismo valor de predicción, dado por la media de los valores de las observaciones del conjunto de entrenamiento que caigan en esa región.

Ahora nos podemos preguntar cómo es el proceso de división del espacio en regiones disjuntas. En teoría, las regiones no tienen ninguna forma específica. Sin embargo, los algoritmos existentes dividen el espacio de las variables predictoras en rectángulos o “cajas” para simplificar la interpretación del modelo predictivo obtenido. El objetivo es encontrar las regiones que minimicen el RRS (Suma de Cuadrados de los Residuos), dado por la expresión siguiente:

$$\sum_{j=1}^{J}\sum_{i\in R_j}(y_i-\hat{y}_{R_j})^2$$

donde $\hat{y}_{R_j}$ es la media de los valores de las observaciones del conjunto de entrenamiento que están en la región j. Desafortunadamente, no es factible computacionalmente considerar todas las posibles particiones del espacio predictor en J regiones. Debido a esto, se utiliza una metodología de arriba hacia abajo conocido como división binaria recursiva. El enfoque es de arriba hacia abajo porque comienza en la parte superior del árbol (en ese punto todas las observaciones pertenecen a una sola región) y luego divide sucesivamente el espacio predictor; cada división se indica a través de dos nuevas ramas más abajo en el árbol. Se puede decir que es codicioso porque en cada paso del proceso de construcción del árbol, la mejor división se realiza en ese paso particular, en lugar de mirar hacia más allá y elegir una división que conducirá a un mejor árbol en un paso futuro.

Para empezar a realizar la división binaria recursiva, primero seleccionamos un predictor X_j y un punto de corte s que divida el espacio de las variables predictoras en dos regiones $\{X|X_j < s\}$ y $\{X|X_j \geq s\}$ de forma que se reduzca lo máximo posible el RSS. Esto es, consideramos todas las variables predictoras $X_1, \dots, X_p$ y todos los posibles valores de corte s para cada predictor. Tras esto, elegimos el predictor y el punto de corte con los que obtenemos un menor RSS. De manera más específica, consideramos los siguientes pares de hiperplanos:

$R_1(j,s) = \{X|X_j < s\}$ y $R_2(j,s) = \{X|X_j \geq s\}$

Ahora habría que buscar los valores de j y s que minimicen la expresión siguiente:

$$\sum_{i\in R_1(j,s)}(y_i-\hat{y}_{R_1})^2+\sum_{i\in R_2(j,s)}(y_i-\hat{y}_{R_2})^2$$

Encontrar los valores (j,s) que minimicen la expresión anterior no es un proceso que requiera mucho tiempo, sobre todo si el conjunto de variables predictoras no es muy grande.

Tras hacer una primera división, repetimos el proceso, buscando el mejor predictor y el mejor punto de corte para dividir aún más los datos con el fin de minimizar el RSS dentro de cada una de las regiones resultantes. Sin embargo, esta vez, como la partición con menor RSS la vamos a encontrar en una de las dos regiones, en lugar de dividir todo el espacio del predictor, dividimos una de las dos regiones previamente identificadas. Ahora tenemos tres regiones. Una vez más, buscamos dividir una de estas tres regiones más, para minimizar el RSS. El proceso continúa

hasta que se alcanza un criterio de detención; por ejemplo, podemos continuar hasta que ninguna región contenga más de cinco observaciones.

Una vez que tengamos creadas las regiones $R_1, \dots, R_J$, predecimos el valor de una observación dada mediante la media de los valores de las observaciones del conjunto de entrenamiento que estén en la región a la que pertenece la observación dada.

a. Poda del árbol

El proceso descrito anteriormente puede producir buenas predicciones en el conjunto de entrenamiento, pero es probable que sobreajuste los datos, lo que conduce a un rendimiento pobre sobre el conjunto test. Esto se debe a que el árbol resultante puede ser demasiado complejo. Un árbol más pequeño con menos divisiones, es decir, con menos regiones $R_1, \dots, R_J$, puede conducir a una menor varianza y una mejor interpretación a costa de un pequeño sesgo. Una posible alternativa al proceso descrito anteriormente es construir el árbol siempre que la disminución en el RSS debido a cada división exceda algún umbral. Esta estrategia dará como resultado árboles más pequeños, pero sigue teniendo un problema, ya que una división aparentemente inútil al comienzo del árbol podría estar seguida por una muy buena división, es decir, una división que conduce a una gran reducción en RSS.

Por lo tanto, una mejor estrategia es construir un árbol muy grande T_0, y luego podarlo para obtener un árbol secundario. Intuitivamente, nuestro objetivo es seleccionar un subárbol que nos conduzca al error de test más bajo. Dado un subárbol, podemos estimar su error de test utilizando la validación cruzada o un conjunto de validación. Sin embargo, estimar el error de test mediante validación cruzada para cada subárbol posible sería demasiado engorroso, ya que existe una cantidad muy grande de subárboles posibles. Por tanto, necesitamos una forma de seleccionar un pequeño conjunto de subárboles para su consideración.

Se define para ello un criterio coste-complejidad. En lugar de considerar todos los subárboles posibles, consideramos una secuencia de árboles indexada por un parámetro de ajuste no negativo α.

A cada valor de α le corresponde un subárbol $T \subset T_0$ tal que la expresión

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} \left(y_i - \hat{y}_{R_m}\right)^2 + \alpha |T|$$

sea lo más pequeña posible. En la expresión anterior, $|T|$ indica el número de nodos terminales del árbol T, R_m es la región correspondiente al m-ésimo nodo terminal de T, y $\hat{y}_{R_m}$ es el valor de predicción en la región R_m.

El parámetro de ajuste α controla una compensación entre la complejidad o tamaño del subárbol y su ajuste a los datos de entrenamiento. Cuando $\alpha = 0$, entonces el subárbol T será igual a T_0, porque entonces $\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$ simplemente mide el error de entrenamiento. Sin embargo, a medida que α aumenta, hay que pagar un precio por tener un árbol con muchos nodos terminales, por lo que la cantidad $\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$ tenderá a minimizarse para un subárbol más pequeño.

A medida que aumenta α, las ramas del árbol se podan de forma anidada y predecible, por lo que es fácil obtener la secuencia completa de subárboles en función de α. Podemos seleccionar un valor de α usando un conjunto de validación o usando validación cruzada. Luego volvemos al conjunto completo de datos y obtenemos el subárbol correspondiente a α. Este proceso se resume en el algoritmo siguiente:

1- Utilizar la división binaria recursiva para construir un árbol grande con los datos de entrenamiento, deteniéndose solo cuando cada nodo terminal tenga menos de un número mínimo de observaciones.
2- Aplicar la poda de complejidad de costos al árbol grande para obtener una secuencia de los mejores subárboles, como función de α.
3- Usar la validación cruzada para elegir α. Es decir, dividir las observaciones de entrenamiento en K subconjuntos. Para cada $k = 1, \dots, K$:
    (a) Repetir los pasos 1 y 2 en todos menos en el k-ésimo subconjunto de entrenamiento.
    (b) Evaluar el error de predicción cuadrático medio en los datos del k-ésimo subconjunto como función de α.

    Realizar la media de los resultados para cada valor de α, y elegir α para minimizar el error promedio.
4- El subárbol elegido es el correspondiente del paso 2 con el valor de α elegido

Árboles de clasificación

Los árboles de clasificación son muy similares a los de regresión, con la diferencia que ahora la respuesta a predecir es cualitativa en lugar de cuantitativa. En este caso, en lugar de predecir la respuesta de una observación dada mediante la media de las observaciones del conjunto de

entrenamiento que pertenecen a la región en la que cae dicha observación, se toma como respuesta la moda de las observaciones que pertenecen a la región. Al interpretar los resultados de un árbol de clasificación, a menudo nos interesa no solo la predicción de la clase correspondiente a una región dada por un nodo terminal, sino también las proporciones de las distintas clases entre las observaciones de entrenamiento que caen dentro de esa región.

La construcción de un árbol de clasificación es bastante similar a la de un árbol de regresión. Al igual que en el caso de regresión, utilizamos la división binaria recursiva para construir un árbol de clasificación. Sin embargo, en el caso de la clasificación, RSS no se puede utilizar como criterio para realizar las divisiones binarias. Una alternativa natural a RSS es la tasa de error de clasificación. Al asignar una observación en una región determinada a la clase más común de observaciones de entrenamiento en esa región, la tasa de error de clasificación es simplemente la fracción de las observaciones de entrenamiento en esa región que no pertenecen a la clase más común:

$$E = 1 - max_k(\hat{p}_{mk})$$

En la expresión anterior, $\hat{p}_{mk}$ representa la proporción de observaciones del conjunto de entrenamiento que pertenecen a la región m-ésima y a la clase k-ésima. Sin embargo, la tasa de error de clasificación no es un criterio que funcione muy bien. Por ello, se buscan otras opciones más fiables.

El índice de Gini viene definido por una medida de la varianza total en las K clases:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Viendo la expresión, sabemos que toma un valor pequeño si todos los $\hat{p}_{mk}$ $k = 1, \dots, K$ están cercanos a 0 ó a 1. Por esta razón, se hace referencia al índice de Gini como una medida de la pureza del nodo: un valor pequeño indica que un nodo contiene predominantemente observaciones de una sola clase.

5. BAGGING Y RANDOM FOREST

Los árboles de decisión suelen ser inestables, es decir, pequeños cambios en el conjunto de entrenamiento conducen a modelos muy distintos. Para reducir esta inestabilidad, y en general mejorar la capacidad de generalización, se puede recurrir al Bagging y a los modelos Random Forest.

a. Bagging

Para presentar el Bagging empezaremos hablando del bootstrap, un método de remuestreo de gran utilidad. Se suele usar, por ejemplo, en muchas situaciones en las que es difícil o incluso imposible calcular directamente la desviación estándar de una cantidad de interés o bien para calcular intervalos de confianza. El método boostrap se basa en la generación de muestras extraídas de la muestra original, con reemplazmiento y con el mismo tamaño (es lo que se conoce como muestras bootstrap).

Los árboles de decisión, por lo general, sufren de alta varianza. Esto significa que, si dividimos los datos de entrenamiento en dos partes al azar, y adaptamos un árbol de decisión a ambas mitades, los resultados que obtengamos podrían ser bastante diferentes. En cambio, un procedimiento con baja varianza arrojará resultados similares si se aplica repetidamente a conjuntos de datos distintos. El Bagging es un procedimiento cuyo propósito general es reducir la varianza de un método de aprendizaje estadístico.

Supongamos que tenemos un conjunto de n observaciones independientes $Z_1, \dots, Z_n$, cada una de ellas con varianza σ^2. La varianza de la media $\bar{Z}$ de las observaciones viene dada por σ^2/n. En otras palabras, promediar un conjunto de observaciones reduce la varianza. De este modo, una forma reducir la varianza y, así, incrementar el poder de predicción de un método de aprendizaje estadístico es tomar muchos conjuntos de entrenamiento de la población total, construir un modelo predictor sobre cada uno de ellos y realizar la media de los distintos valores predichos. Más específicamente, se trataría de calcular $\hat{f}^1(x), \hat{f}^2(x), \dots, \hat{f}^B(x)$ en el caso de que utilicemos B conjuntos de entrenamiento distintos y realizar la media de todos ellos:

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x)$$

Como no vamos a tener distintas muestras de entrenamiento, tenemos que hacer uso del Bootstrap para generar diferentes muestras de nuestro conjunto de entrenamiento. En este

sentido, se generarán B muestras Bootstrap de nuestro conjunto de entrenamiento y con cada una de ellas se obtendrá una predicción $\hat{f}^{*b}(x)$. El predictor final será la media de todos ellos:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$$

A esto es lo que llamamos Bagging.

Particularmente, el Bagging funciona muy bien en el caso de los árboles de decisión. Se ha demostrado que ofrece mejoras impresionantes en precisión combinando cientos o incluso miles de árboles en un solo procedimiento.

Hasta ahora, hemos descrito el procedimiento de Bagging en el contexto de regresión, para predecir un resultado cuantitativo Y. Para extender el Bagging a un problema de clasificación, donde Y es cualitativo, hay algunos enfoques posibles, pero el más simple es el siguiente. Para una observación dada, podemos registrar la clase predicha por cada uno de los árboles B, y obtener un voto mayoritario, es decir, la predicción general es la clase mayoritaria más común entre las predicciones B.

Por otro lado, hay que decir que el número de árboles B no es un parámetro crítico para el Bagging, es decir, usar un número elevado de árboles no conduce al sobreajuste de los datos de entrenamiento. En la práctica, se suele usar un valor de B donde, a partir de él, el error no se reduce demasiado.

Error “Out Of Bag” (OOB Error)

La estructura de las muestras bootstrap puede ser aprovechada para obtener una estimación de la capacidad de generalización del modelo final, modelo de Bagging, sin la necesidad de llevar a cabo la validación cruzada o de disponer de un conjunto test. Como hemos visto, la clave del Bagging es que los árboles se ajustan repetidamente a distintos subconjuntos del conjunto de entrenamiento, dados por el bootstrap. Se puede demostrar que, en promedio, la muestra bootstrap en la que se basa cada árbol consta de dos tercios de las observaciones. El tercio restante que no se utilizan para ajustar el árbol son llamadas observaciones “out of bag” (OOB). Se puede predecir la respuesta para la observación i-ésima usando cada uno de los árboles en los que esa observación pertenece a OOB. Por tanto, habrá entorno a $B/3$ predicciones para la

observación i -ésima. Para obtener una predicción de la i -ésima observación, podemos promediar todas las respuestas predichas (en caso de estar frente a un problema de regresión) o tomar la moda de los resultados predichos (en caso de estar frente a un problema de clasificación). Esto último es lo que llamaremos una predicción OOB. Obviamente, podemos obtener una predicción OOB para cada una de las n observaciones y calcular una estimación del Error cuadrático medio (para regresión) o una estimación del Error de clasificación (para clasificación). Decimos que el error OOB se trata de una estimación válida para el Error de test debido a que las respuestas obtenidas para cada observación no han utilizado esa observación para crear el modelo.

Importancia de cada variable

Como ya hemos comentado, el Bagging, generalmente, da como resultado una precisión mejorada con respecto a la predicción de un solo árbol. Desafortunadamente, puede ser difícil interpretar el modelo resultante. Una de las ventajas de los árboles de decisión es el diagrama atractivo y fácil de interpretar que se obtiene. Sin embargo, cuando combinamos una gran cantidad de árboles, ya no es posible representar el procedimiento de aprendizaje estadístico resultante utilizando un solo árbol, y ya no está claro qué variables son más importantes para el procedimiento. Por lo tanto, el Bagging mejora la precisión de predicción, pero empeora o reduce la interpretabilidad.

Aunque la combinación de árboles es mucho más difícil de interpretar que un solo árbol, se puede obtener un resumen general de la importancia de cada predictor usando RSS (para árboles de regresión) o el índice de Gini (para árboles de clasificación). En el caso de los árboles de regresión, podemos registrar la cantidad total que el RSS se reduce debido a las divisiones sobre un predictor dado, promediado sobre todos los árboles B. Un valor grande indica que se trata de un predictor importante. De manera similar, en el contexto de los árboles de clasificación, podemos sumar la cantidad total que el índice de Gini se reduce por divisiones sobre un predictor dado, promediado sobre todos los árboles B.

b. Random Forest

Random Forests o, en español, los “Bosques aleatorios” comparte los elementos principales del Bagging, por lo que también se basa en la construcción de un número elevado de árboles de decisión sobre muestras bootstrap. Pero al construir estos árboles de decisión, cada vez que se

considera una división, se elige una muestra aleatoria de m predictores como candidatos obtenidos del conjunto completo de predictores p.

Una nueva muestra de m predictores es tomada en cada división y m suele venir dado por $\sqrt{p}$. Es decir, en el caso de que tengamos 16 variables predictoras, la muestra de predictores que consideremos en cada división sería de 4.

Por tanto, al construir un bosque aleatorio, en cada división del árbol, el algoritmo ni siquiera tiene en cuenta la mayoría de los predictores disponibles. Esto puede parecer una locura, pero tiene un razonamiento ingenioso. Supongamos que hay un predictor muy fuerte en el conjunto de datos, junto con una serie de otros predictores moderadamente fuertes. Entonces, en todos los árboles obtenidos mediante Bagging o en la mayoría de ellos se utilizaría este predictor principal en la división inicial. Como consecuencia, todos los árboles tendrían un aspecto similar, es decir, estaría muy correlacionados. Desafortunadamente, promediar estos árboles tan parecidos no reduciría la varianza tanto como se desea. En particular, esto nos muestra que el Bagging no conduciría a una reducción de la varianza aceptable.

Los bosques aleatorios superan este problema forzando a cada división a considerar solo un subconjunto de m predictores. Por lo tanto, en promedio $(p - m)/p$ de las divisiones ni siquiera considerarán el fuerte predictor, por lo que otros predictores tendrán más posibilidades. Podemos pensar que este proceso reduce la correlación entre los árboles, haciendo que el promedio de los árboles resultantes sea menos variable y, por lo tanto, de confianza. Dicho esto, se puede considerar al Bagging como un Bosque aleatorio con $m = p$.

Como pasa con el Bagging, Random Forests no tiene el problema del sobreajuste de los datos del conjunto de entrenamiento si aumentamos el número de árboles B.

6. BOOSTING

c. Breve introducción

El Boosting es un método de combinación de modelos muy reciente, ya que fue introducido en el año 1990 por Schapire. Es un método de aprendizaje supervisado diseñado originalmente para problemas de clasificación, aunque también puede ser extendido a problemas de regresión obteniendo buenos resultados.

Es una de las ideas de aprendizaje más poderosas introducidas en los últimos veinte años. La idea principal del Boosting es la de crear un clasificador competente combinando muchos clasificadores "débiles" (árboles simples en nuestro caso). En este sentido, el Boosting se asemeja bastante al Bagging aunque, veremos que está relación es meramente superficial y que, realmente, hay diferencias notables entre ambos métodos de aprendizaje.

Principales características:

- Técnica enfocada a mejorar la capacidad predictiva de métodos de aprendizaje estadístico con alta variabilidad (por ejemplo, árboles)
- Los clasificadores se construyen secuencialmente. Cada clasificador se construye usando información del clasificador construido previamente.
- El conjunto de entrenamiento para la construcción de cada clasificador se obtiene muestreando de forma selectiva.
- Las instancias mal clasificadas en una iteración tienen más probabilidad de ser seleccionadas en la siguiente.
- La construcción de cada clasificador depende fuertemente de los clasificadores que han sido construidos previamente.
- Tiene tendencia al sobreajuste.

El Boosting se sustenta en las ideas de que ningún clasificador es siempre el mejor y que la combinación de "clasificadores débiles" puede suavizar las carencias de cada uno, dando lugar a un clasificador más eficiente.

d. Procedimiento Boosting original

Como hemos mencionado anteriormente, el primer procedimiento Boosting fue establecido en 1990 por Schapire. Este método se basaba en seguir los siguientes pasos:

1- Extraer un subconjunto E1 sin reemplazamiento de la muestra de aprendizaje y entrenar un clasificador C1 con este subconjunto.
2- Extraer un segundo subconjunto E2 sin reemplazamiento de la muestra de aprendizaje y añadir el 50% de las instancias mal clasificadas anteriormente. Utilizar este nuevo subconjunto para entrenar un clasificador C2.
3- Encontrar un subconjunto E3 de instancias de la muestra de aprendizaje en las que difieren los clasificadores C1 y C2. Utilizar E3 para entrenar un clasificador C3.
4- Combinar los clasificadores C1, C2 y C3 mediante votación por mayoría.

e. Adaboost

Adaboost fue el primer algoritmo tipo Boosting que tuvo éxito. Fue introducido por Freund y Schapire, y su nombre proviene de la expresión “Adaptive Boosting”.

Consideramos un problema de dos clases, con la variable respuesta dada por Y perteneciente a $\{-1,1\}$. La tasa de error del conjunto entrenamiento viene dada por la siguiente expresión:

$$err = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq G(x_i))$$

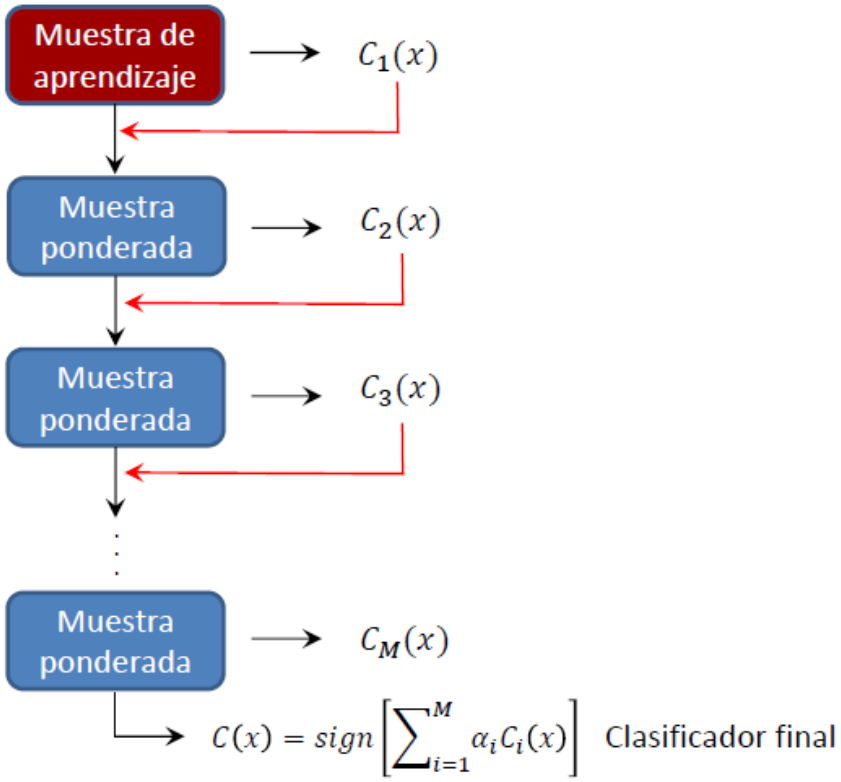
Definimos un clasificador débil como aquel en el que la tasa de error es algo menor al 50% en el caso en que tengamos dos clases, y algo menor al $\frac{K-1}{K}100\ \%$ en el caso de tener K clases, es decir, aquel que mejora ligeramente a clasificar los distintos registros “lanzando una moneda o un dado de K caras”. El Boosting se encarga de aplicar secuencialmente un clasificador débil a versiones modificadas de los datos, produciendo una secuencia de clasificadores débiles $G_m(x), m = 1,2, \dots, M$.

La predicción final viene dada por una combinación ponderada de todos los clasificadores débiles que se han producido anteriormente:

$$G(x) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

donde $\alpha_1, \alpha_2, \ldots, \alpha_M$ son los pesos de cada clasificador débil, que posteriormente veremos cómo se calculan. La función de estos pesos es dar mayor importancia o influencia a los clasificadores con mayor tasa de acierto o menor tasa de error en la secuencia.

La figura siguiente describe de una forma muy intuitiva todo el proceso anterior:



Cuando hemos dicho anteriormente "versiones modificadas de los datos" en cada paso, nos referíamos a que se aplican distintos pesos $w_1, w_2, \ldots, w_N$ a cada observación de la muestra de entrenamiento. En el primer paso todas las observaciones tienen el mismo peso:

$$w_i = \frac{1}{N}, \qquad i = 1,2, \ldots, N$$

A partir de aquí, en cada iteración posterior, los pesos o ponderaciones son modificadas en cada una de las observaciones y el algoritmo se aplica a las observaciones ponderadas. En la iteración m, las observaciones que estén mal clasificadas por el clasificador del paso $m-1$, se ven afectadas por un incremento de sus ponderaciones. En cambio, las observaciones bien clasificadas por el clasificador anterior disminuyen su ponderación. Es decir, en la iteración m, tendrán más peso las observaciones que han sido mal clasificadas por el clasificador $m-1$. De

este modo, cada clasificador es forzado a concentrarse principalmente en las observaciones mal clasificadas por clasificadores previos en la secuencia.

El algoritmo Adaboost se compone de los siguientes pasos:

1- Inicialmente, todas las observaciones tienen el mismo peso. Es decir,

$$w_i = \frac{1}{N}, \qquad i = 1,2,\dots,N$$

2- Desde $m = 1$ hasta $m = M$:

a. Ajustar un clasificador G al conjunto de entrenamiento usando los pesos w_i.

b. Calculamos el error err_m teniendo en cuenta los pesos de las observaciones calculados en la iteración anterior:

$$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G(x_i))}{\sum_{i=1}^{N} w_i}$$

c. Calculamos el peso o ponderación del clasificador:

$$\alpha_m = \log\left(\frac{1 - err_m}{err_m}\right)$$

d. Actualizamos los pesos de las observaciones para la iteración siguiente y los escalamos:

$$w_i \leftarrow w_i \, e^{\alpha_m I(y_i \neq G(x_i))}$$

3- El clasificador final viene dado por la expresión:

$$G(x) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

Fijémonos que las observaciones mal clasificadas multiplican su peso por e^{α_m} y las observaciones bien clasificadas por $e^{-\alpha_m}$. Como α_m es positivo (suponiendo que $err_m <$

$0.5\ \forall m$), confirmamos lo que habíamos comentado antes: las observaciones mal clasificadas aumentan su peso en la siguiente iteración y las que están bien clasificadas lo disminuyen.

A diferencia que el algoritmo de Boosting original, Adaboost utiliza toda la muestra de entrenamiento para entrenar a cada clasificador.

El algoritmo que acabamos de presentar es conocido como "Adaboost Discreto" debido a que el clasificador base devuelve la predicción de la clase a la que pertenece cada elemento. En cambio, si el clasificador base devolviera un valor real, el algoritmo Adaboost podría ser modificado apropiadamente. Esto último es conocido como "Real Adaboost".

f. Boosting se ajusta a un modelo aditivo

La clave del éxito del boosting reside en la expresión

$$G(x) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

Boosting es una manera de ajustar un desarrollo aditivo en un conjunto de funciones base elementales. En este caso, las funciones base son los distintos clasificadores (árboles) de cada iteración $G_m(x)$. Más generalmente, los desarrollos de funciones base tienen la siguiente forma:

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$$

donde $\beta_m,\ m = 1,2,\dots,M$ son los coeficientes del desarrollo y $b(x;\gamma) \in \mathbb{R}$ son, normalmente, funciones simples que dependen de x y de un conjunto de parámetros γ. En el caso de los árboles, los parámetros γ se utilizan para parametrizar las variables separadoras y los puntos separadores en los nodos internos.

Normalmente, estos modelos se ajustan minimizando una función de pérdida promediada sobre el conjunto de entrenamiento, como el error cuadrático o una función de pérdida basada en la verosimilitud.

$$min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)\right)$$

Para muchas funciones de pérdida y/o funciones base, esto requiere técnicas de optimización numéricas intensivas computacionalmente. Sin embargo, una alternativa simple que puede ser llevada a cabo a menudo cuando es factible resolver rápidamente el subproblema de ajustar únicamente una función base simple:

$$min_{\{\beta,\gamma\}} \sum_{i=1}^{N} L\big(y_i, b(x_i;\gamma_m)\big)$$

g. Modelo aditivo por pasos hacia adelante

Mediante este método se aproxima la solución de

$$min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)\right)$$

añadiendo secuencialmente nuevas funciones base a la expansión sin ajustar los parámetros y los coeficientes de las que ya se han agregado. El método sería el siguiente:

1- Establecer $f_0(x) = 0$

2- Desde $m = 1$ a $m = M$:

a- Calcular

$$(\beta_m, \gamma_m) = \arg min_{\{\beta,\gamma\}} \sum_{i=1}^{N} L\big(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)\big)$$

b- Establecer

$$f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma_m)$$

Para la función de pérdida basada en el error cuadrático

$$L\big(y, f(x)\big) = \big(y - f(x)\big)^2$$

tenemos lo siguiente:

$$L\big(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)\big) = \big(y_i - f_{m-1}(x_i) - \beta b(x_i;\gamma)\big)^2 = \big(r_{im} - \beta b(x_i;\gamma)\big)^2$$

donde $r_{im} = y_i - f_{m-1}(x_i)$ es el residuo del modelo actual en la i-ésima observación.

De este modo, el término $\beta_m b(x_i; \gamma_m)$ que mejor se adapte a los residuos en cada etapa es añadido al desarrollo o suma en cada paso. Sin embargo, como mostraremos luego, la función de pérdida basada en el error cuadrático generalmente no es una buena opción para la clasificación; por lo tanto, aparece la necesidad de considerar otras funciones de pérdida.

h. Función de pérdida exponencial y adaboost

Ahora mostraremos que el algoritmo AdaBoost expuesto anteriormente (AdaBoost.M1) es equivalente a un modelo aditivo por pasos hacia adelante usando la siguiente función de pérdida:

$$L\big(y, f(x)\big) = e^{-yf(x)}$$

Para AdaBoost, las funciones base son los diferentes clasificadores individuales $G_m(x) \in \{-1,1\}$. Usando la función de pérdida presentada (función de pérdida exponencial), se debe resolver

$$(\beta_m, G_m) = \arg min_{\{\beta,G\}} \sum_{i=1}^{N} e^{-y_i(f_{m-1}(x_i)+\beta G(x_i))}$$

$$= \arg min_{\{\beta,G\}} \sum_{i=1}^{N} w_i^{(m)} e^{-y_i(\beta G(x_i))}$$

con $w_i^{(m)} = e^{-y_i f_{m-1}(x_i)}$

Como $w_i^{(m)}$ no depende ni de β ni de $G(x)$, puede ser considerado como el peso aplicado a cada observación. Este peso depende de $f_{m-1}(x_i)$, por lo que el peso irá cambiando en cada iteración.

La solución a

$$(\beta_m, G_m) = \arg min_{\{\beta,G\}} \sum_{i=1}^{N} w_i^{(m)} e^{-y_i(\beta G(x_i))}$$

puede ser obtenida en dos pasos:

1- Para cualquier valor de $\beta > 0$, la solución para $G_m(x)$ viene dada por:

$$G_m = \arg min_G \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i))$$

que es el clasificador que minimiza la tasa ponderada de error.

2- Para hallar la solución del parámetro β escribimos la expresión $\sum_{i=1}^{N} w_i^{(m)} e^{-y_i(\beta G(x_i))}$ como sigue:

$$\sum_{i=1}^{N} w_i^{(m)} e^{-y_i(\beta G(x_i))} = e^{-\beta} \sum_{y_i = G(x_i)} w_i^{(m)} + e^{\beta} \sum_{y_i \neq G(x_i)} w_i^{(m)}$$

$$= \left(e^{\beta} - e^{-\beta}\right) \sum_{i=1}^{N} w_i^{(m)} I\left(y_i \neq G(x_i)\right) + e^{-\beta} \sum_{i=1}^{N} w_i^{(m)}$$

Sustituimos $G(x)$ por $G_m(x)$ (clasificador que nos da la tasa de error mínima) y resolvemos para β (suponiendo pesos normalizados):

$$\left(e^{\beta} - e^{-\beta}\right) \sum_{i=1}^{N} w_i^{(m)} I\left(y_i \neq G(x_i)\right) + e^{-\beta} \sum_{i=1}^{N} w_i^{(m)} = e^{\beta} err_m + e^{-\beta}(1 - err_m)$$

Aplicamos logaritmos y derivamos para obtener el β con el que se hace mínimo:

$$\log(e^{\beta} err_m + e^{-\beta}(1 - err_m)) \rightarrow derivamos\ con\ respecto\ a\ \beta\ e\ igualamos\ a\ 0$$

$$\frac{1}{e^{\beta} err_m + e^{-\beta}(1 - err_m)} e^{\beta} err_m - e^{-\beta}(1 - err_m) = 0$$

$$\rightarrow e^{\beta} err_m - e^{-\beta}(1 - err_m) = 0$$

$$\rightarrow e^{\beta} err_m = e^{-\beta}(1 - err_m)$$

$$\rightarrow \beta + \log(err_m) = -\beta + \log(1 - err_m)$$

$$\rightarrow 2\beta = \log(1 - err_m) - \log(err_m)$$

$$\rightarrow \beta = \frac{1}{2} \log\left(\frac{1 - err_m}{err_m}\right)$$

Una vez hemos hallado la solución de ambos parámetros, actualizamos $f_m(x) = f_{m-1}(x) + \beta_m G_m(x)$

Los pesos para la siguiente iteración vienen dados por $w_i^{(m+1)} = e^{-y_i f_m(x_i)} = w_i^{(m)}\, e^{-y_i \beta_m G_m(x_i)}$

Usando que $-y_i G_m(x_i) = 2\, I\big(y_i \neq G_m(x_i)\big) - 1$, entonces:

$$w_i^{(m+1)} = w_i^{(m)} e^{\alpha_m I(y_i \neq G_m(x_i)} e^{-\beta_m}$$

Donde $\alpha_m = 2\beta_m$

El factor $e^{-\beta_m}$ multiplica a todos los pesos por el mismo valor, por tanto, no es necesario y se puede eliminar. Así, obtenemos una expresión equivalente a la que dábamos cuando explicamos el algoritmo AdaBoost.M1. Con esto, podemos concluir que AdaBoost.M1 minimiza una función de pérdida exponencial mediante un modelo aditivo por pasos hacia adelante.

i. ¿Por qué función de pérdida exponencial?

El algoritmo AdaBoost.M1 procede de una perspectiva completamente diferente a la presentada en la sección anterior. La equivalencia del algoritmo con el modelo aditivo por pasos hacia delante utilizando una función de pérdida exponencial fue descubierta 5 años después. Estudiando las propiedades de la función de pérdida exponencial, podemos obtener información acerca del procedimiento y descubrir formas en las que se puede mejorar.

La principal atracción del uso de una función de pérdida exponencial es computacional, ya que conduce al simple algoritmo AdaBoost de reponderación modular. Sin embargo, hay que preguntarse sobre sus propiedades estadísticas, es decir, que estima y si lo que estima está bien estimado. En cuanto a la primera cuestión, podemos decir que se responde buscando su minimizador de población.

Es fácil demostrar que

$$f^*(x) = \arg min_{f(x)}\, E_{Y|X}\big(e^{-Yf(x)}\big) = \frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)}$$

Es decir, utilizando que $\Pr(Y=1|x) = 1 - \Pr(Y=-1|x)$ obtenemos que:

$$\Pr(Y = 1|x) = \frac{1}{1 + e^{-2f^*(x)}}$$

De este modo, la expansión aditiva producida por AdaBoost está estimando el log-ratio de $\Pr(Y = 1|x)$. Por eso se usa su signo como regla de clasificación.

Otro criterio de pérdida con el mismo minimizador de la población es la log-verosimilitud de la binomial negativa o la devianza.

Sea

$$p(x) = \Pr(Y = 1|x) = \frac{e^{f(x)}}{e^{-f(x)} + e^{f(x)}} = \frac{1}{1 + e^{-2f^*(x)}}$$

Definimos $Y' = \frac{Y+1}{2} \in \{0,1\}$. Entonces, la función de pérdida basada en la log-verosimilitud de la binomial viene dada por:

$$l\big(Y, p(x)\big) = \mathrm{Y}' \log p(x) + (1 - Y') \log(1 - p(x))$$

Equivalentemente la devianza es:

$$-l\big(Y, f(x)\big) = \log(1 + e^{-2Yf(x)})$$

j. Funciones de pérdida y robustez

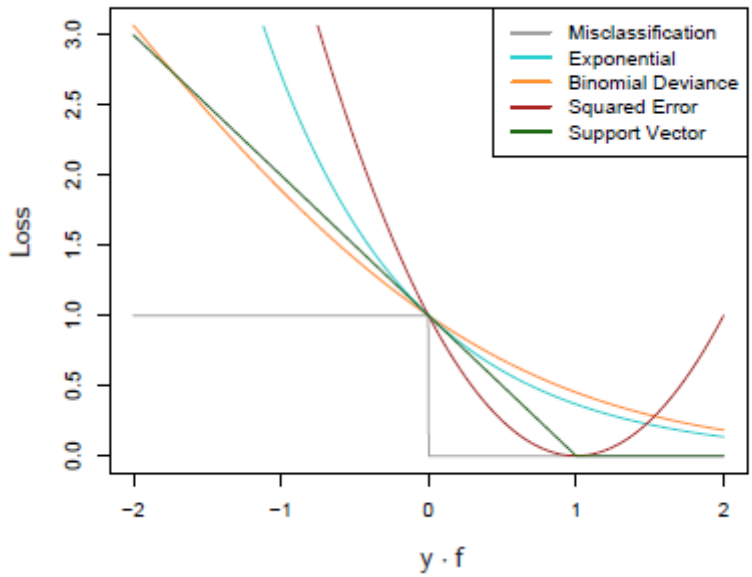
Ahora examinaremos diferentes funciones de pérdida para clasificación y regresión de manera más exhaustiva. Las clasificaremos en términos de su robustez frente a datos extremos.

Funciones de pérdida robustas para clasificación

A pesar de que el rendimiento tanto de la función de pérdida exponencial como de la función de pérdida basada en la devianza tienen la misma solución cuando se aplica a la distribución conjunta de la población, lo mismo no es cierto para conjuntos de datos finitos. Ambas funciones son monótonas decrecientes en función de $yf(x)$. Para clasificación, en el caso de que tengamos dos opciones como variable respuesta (+1 y -1), el término $yf(x)$ juega un papel análogo al del residuo $y - f(x)$ en regresión. La regla de clasificación establecida ($G(x) = sign\,(f(x))$) implica que las observaciones en las que $y_i f(x_i) > 0$ están correctamente

clasificadas mientras que, en cambio, las observaciones donde $y_i f(x_i) < 0$ están mal clasificadas.

Dicho esto, el objetivo de un algoritmo de clasificación viene dado por producir términos $y_i f(x_i)$ (márgenes) positivos en el mayor número de observaciones posible. Por tanto, cualquier criterio (función) de pérdida usado para clasificación debería penalizar en mayor medida los márgenes negativos.



En la imagen anterior se puede ver la variación de distintas funciones de pérdida en función del margen $yf(x)$. Podemos ver como las funciones de pérdida exponencial y devianza penalizan en mayor medida los márgenes negativos de manera continua. La diferencia entre ambas está en el grado. A medida que el margen crece negativamente, la devianza crece linealmente mientras que la función de pérdida exponencial lo hace exponencialmente.

El criterio exponencial concentra mucha más influencia en las observaciones con márgenes negativos. De hecho, conforme menor sea el margen, mayor será la diferencia entre el criterio exponencial y los demás. La devianza concentra relativamente menos influencia en tales observaciones, distribuyendo más uniformemente la influencia entre todos los datos. Por lo tanto, es mucho más robusta en entornos ruidosos donde la tasa de error de Bayes no es cercana a cero, y especialmente en situaciones donde hay una especificación incorrecta de las etiquetas

de clase en los datos de entrenamiento. El rendimiento de AdaBoost se ha observado empíricamente que se degrada dramáticamente en tales situaciones.

Vemos también mostrado en la figura la función de pérdida basada en el error cuadrático. $f^*(x)$ viene dado ahora por:

$$f^*(x) = \arg min_{f(x)} E_{Y_{|x}}\big(Y - f(x)\big)^2 = E(Y|x) = 2\Pr(Y = 1|x) - 1$$

Recordemos que la regla de clasificación viene dada por $G(x) = sign\,(f(x))$. La función de pérdida basada en el error cuadrático no es un buen sustituto del error de clasificación debido a que, como se puede observar en la gráfica, no es monótona decreciente con respecto al margen $yf(x)$. Para los márgenes con valor $y_i f(x_i) > 1$, está función de pérdida se incrementa cuadráticamente, otorgando mayor influencia a observaciones que están bien clasificadas. De este modo, la influencia relativa de las observaciones mal clasificadas se reduciría. Por tanto, una función de pérdida monótona decreciente con respecto a $yf(x)$ puede ser vista como el mejor sustituto del error de clasificación.

En el caso de tener una clasificación de K clases, la variable respuesta Y toma valores en el conjunto $G = \{G_1, \dots, G_K\}$. Por tanto, ahora habría que buscar un clasificador $G(x)$ que tome valores en G. Para ello, es suficiente conocer las probabilidades condicionadas de pertenencia a cada clase $p_k(x) = \Pr(Y = G_k|x)\,, k = 1, \dots, K$. De este modo,

$$G(x) = G_k \; donde \; k = \arg\max_l p_l(x)$$

La probabilidad condicionada a cada clase viene dada por la siguiente fórmula:

$$p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}$$

De esta forma nos aseguramos de que $0 \leq p_k(x) \leq 1$ y que la suma sea 1. Fijémonos en que ahora tenemos K funciones diferentes. Existe una redundancia en las funciones $f_k(x)$, ya que al agregar una $h(x)$ arbitraria a cada una, el modelo permanece sin cambios. Tradicionalmente, una de ellas la establecemos como 0. Aquí, preferimos retener la simetría e imponer que $\sum_{l=1}^{K} f_l(x) = 0$. La devianza para el problema extendido a K clases viene dada por:

$$L\big(y, p(x)\big) = -\sum_{k=1}^{K} I(y = G_k) \log p_k(x) = -\sum_{k=1}^{K} I(y = G_k)\, f_k(x) + \log \sum_{l=1}^{K} e^{f_l(x)}$$

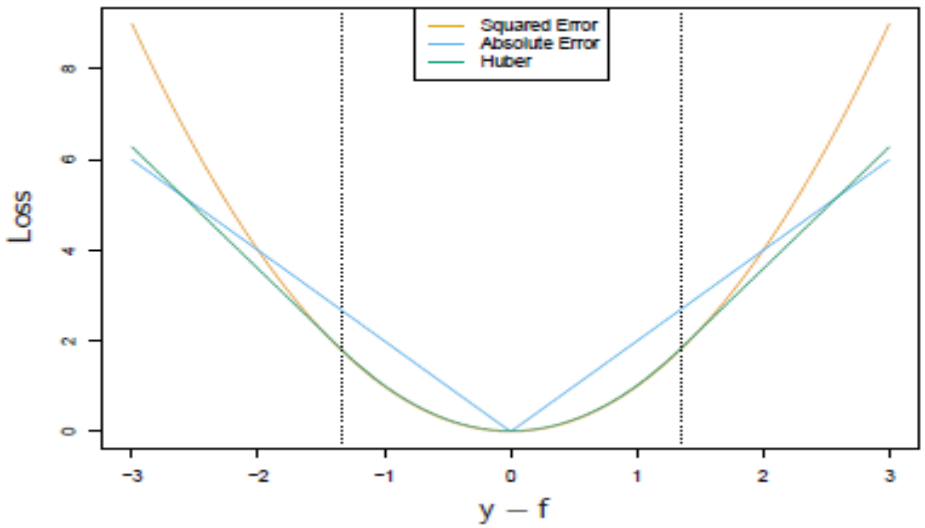
Como en el caso de 2 clases, aquí se penaliza las observaciones mal clasificadas.

Funciones de pérdida robustas para regresión

En el caso de la regresión, la relación entre la función de pérdida basada en el error cuadrático $L(y, f(x)) = (y - f(x))^2$ y la función de pérdida basada en el error absoluto $L(y, f(x)) = |y - f(x)|$ es análoga a la que teníamos para el caso de la clasificación entre las funciones de pérdida exponencial y devianza. Las soluciones que hacen mínimas estás funciones vienen dadas por $f(x) = E(Y|x)$ en el caso de la función basada en el error cuadrático, y $f(x) = mediana(Y|x)$ para la función basada en el valor absoluto. En el caso de muestras finitas, la función de pérdida basada en el error cuadrático pone mucho más énfasis en las observaciones con mayor residuo absoluto $|y_i - f(x_i)|$ durante el proceso de adaptación o entrenamiento. Por lo tanto, es mucho menos robusto y su rendimiento se degrada para las distribuciones de error con mucha varianza y, especialmente, para los valores atípicos (outliers). El criterio de pérdida basado en el valor absoluto funciona mucho mejor en este tipo de situaciones. Además, se han propuesto una variedad de criterios de pérdida que proporcionan una fuerte resistencia (si no una inmunidad absoluta) a valores atípicos brutos mientras que son casi tan eficaces como mínimos cuadrados para errores de Gauss. Uno de ellos es el criterio de pérdida Huber, dado por la siguiente función:

$$L(y, f(x)) = \begin{cases} (y - f(x))^2 \; si \; |y - f(x)| \leq \delta \\ 2\delta|y - f(x)| - \delta^2 \; en \; caso \; contrario \end{cases}$$

En la siguiente figura se muestra una comparación de los tres criterios (funciones) que hemos comentado anteriormente:



Estas consideraciones sugieren que cuando la solidez es una preocupación, como es especialmente el caso en las aplicaciones de minería de datos, el criterio de pérdida del error

cuadrático para la regresión y el criterio de pérdida exponencial para la clasificación no son los mejores desde una perspectiva estadística. Sin embargo, ambos conducen a los elegantes algoritmos modulares Boosting en el contexto del modelado aditivo hacia adelante por etapas. Para el criterio de pérdida del error cuadrático, uno simplemente ajusta el predictor base a los residuos del modelo actual $y_i - f_{m-1}(x_i)$ en cada paso. Para la función de pérdida exponencial, se realiza un ajuste ponderado del clasificador base con los valores de salida y_i, con ponderaciones $w_i = e^{-y_i f_{m-1}(x_i)}$. El uso de otros criterios más sólidos en su lugar no da lugar a algoritmos factibles de Boosting tan simples. Sin embargo, más adelante mostramos cómo se pueden derivar algoritmos de Boosting simples y elegantes basados en cualquier criterio de pérdida diferenciable, lo que produce procedimientos Boosting altamente robustos para la extracción de datos.

k. Diversos procedimientos para la minería de datos

El aprendizaje predictivo es un aspecto importante de la minería de datos. Para cada método para el aprendizaje predictivo en particular, hay situaciones para las que es particularmente adecuado, y otras en las que funciona mal en comparación con lo que se podría hacer con esos datos. Rara vez se sabe de antemano qué procedimiento funcionará mejor o incluso bien para un problema determinado.

Las aplicaciones de minería de datos industriales y comerciales tienden a ser especialmente desafiantes en términos de los requisitos establecidos en los procedimientos de aprendizaje. Los conjuntos de datos a menudo son muy grandes en términos de número de observaciones y cantidad de variables. De este modo, hay que tener en cuenta ciertas consideraciones computacionales. Además, los datos vienen normalmente, en un primer momento, de una forma desordenada (no limpia), la cual da muchos problemas a la hora de realizar distintos métodos. Suele haber variables de todo tipo: cuantitativas, cualitativas (con muchos niveles) y binarias; además de muchos valores perdidos (missing values). Las distribuciones de las variables predictoras numéricas y de la variable respuesta suelen tener mucha varianza y ser muy sesgadas y, además, se miden en escalas muy diferentes y, por lo general, contienen una fracción sustancial de errores de medición (valores atípicos).

En las aplicaciones de minería de datos, generalmente solo una pequeña fracción de la gran cantidad de variables de predicción que se han incluido en el análisis son realmente relevantes para la predicción. Además, las aplicaciones de minería de datos generalmente requieren

modelos interpretables, es decir, no es suficiente simplemente producir predicciones. También es deseable contar con información que brinde una comprensión cualitativa de la relación entre los valores conjuntos de las variables de entrada y el valor de respuesta pronosticado. Por lo tanto, los métodos de "caja negra", como las redes neuronales, que pueden ser bastante útiles en configuraciones puramente predictivas como el reconocimiento de patrones, son mucho menos útiles para minería de datos.

Estos requisitos de velocidad, la capacidad de interpretación y la naturaleza desordenada de los datos limitan agudamente la utilidad de la mayoría de los procedimientos de aprendizaje como métodos estándar para la extracción de datos. Un método "listo para usar" es uno que se puede aplicar directamente a los datos sin requerir una gran cantidad de tiempo de preprocesamiento de datos o ajuste cuidadoso del procedimiento de aprendizaje.

De todos los métodos de aprendizaje bien conocidos, los árboles de decisión son los más cercanos a cumplir con los requisitos para servir como un procedimiento disponible para la minería de datos. Son relativamente rápidos de construir y producen modelos interpretables (si los árboles son pequeños). Naturalmente incorporan mezclas de variables predictivas numéricas y categóricas y valores perdidos. Son invariantes bajo transformaciones (estrictamente monótonas) de los predictores individuales. Como resultado, escalar y/o realizar transformaciones más generales no son un problema, y son inmunes a los efectos de los valores atípicos del predictor. Realizan la selección de características internas como una parte integral del procedimiento. Por lo tanto, son resistentes, si no completamente inmunes, a la inclusión de muchas variables predictivas irrelevantes. Estas propiedades de los árboles de decisión son en gran medida la razón por la que se han convertido en el método de aprendizaje más popular para la minería de datos.

Los árboles tienen un aspecto que les impide ser la herramienta ideal para el aprendizaje predictivo, es decir, la inexactitud. Rara vez brindan una precisión predictiva comparable a la mejor que se puede lograr con los datos disponibles.

Como se vio y se explicó al principio de este documento, aumentar los árboles de decisión mejora su precisión. Al mismo tiempo, mantiene la mayoría de sus propiedades deseables para la minería de datos. Algunas de las ventajas de los árboles que se pierden mediante el Boosting son la velocidad, la capacidad de interpretación y, para AdaBoost, la solidez frente a la superposición de las distribuciones de clases y especialmente el etiquetado incorrecto de los datos de entrenamiento. Un modelo de "Gradient Boosting" (GBM) es una generalización del

aumento de árboles que intenta mitigar estos problemas, a fin de producir un procedimiento comercial preciso y efectivo para la extracción de datos.

I. Modelos Boosting basados en árboles

Los árboles de clasificación y de regresión dividen el espacio que forman las variables predictoras en regiones disjuntas $R_j, j = 1,2, \dots, J$, las cuales son representadas por los nodos del árbol. A cada región R_j se le asigna como valor de predicción una constante γ_j. Es decir,

$$x \in R_j \rightarrow f(x) = \gamma_j$$

De esta manera, un árbol puede ser expresado de la siguiente forma:

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j \, I(x \in R_j)$$

Donde $\Theta = \{R_j, \gamma_j\}, j = 1,2, \dots, J$. Estos parámetros se obtienen mediante minimización de la función de pérdida que consideremos:

$$\widehat{\Theta} = \arg min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j)$$

Debido a que resolver el problema de optimización anterior es muy complicado computacionalmente (problema de optimización combinatorio), nos conformamos aproximando la solución. Para ello, dividimos el problema en dos partes:

- **Dado R_j, establecer γ_j**: Esta parte es trivial, ya que, para problemas de regresión, γ_j viene dado por la media de las observaciones que caen en esa región, es decir, $\widehat{\gamma_j} = \overline{y_j}$. En el caso de que estemos ante un problema de clasificación, $\widehat{\gamma_j}$ viene dado por la moda de las observaciones que caigan en la región.
- **Encontrar R_j:** Está es la parte más difícil sin duda alguna, para la cual se encuentran soluciones aproximadas. Una estrategia típica es utilizar un codicioso algoritmo de partición recursivo descendente para encontrar R_j. Además, a veces es necesario aproximar la expresión $\widehat{\Theta} = \arg min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j)$ mediante un criterio más suave y más conveniente para optimizar R_j:

$$\widetilde{\Theta} = \arg min_{\Theta} \sum_{i=1}^{N} \tilde{L}(y_i, T(x_i, \Theta))$$

Entonces, dado $\widehat{R_J} = \widetilde{R_J}, \gamma_j$ puede ser estimado con mayor precisión usando el criterio original.

Un modelo de Boosting basado en árboles es, básicamente, una suma de árboles:

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_M)$$

Los parámetros $\Theta_m, m = 1,2, \dots, M$ se calculan mediante un proceso de pasos hacia delante (algoritmo explicado anteriormente), es decir, en cada paso debemos resolver el siguiente problema de minimización:

$$\widehat{\Theta}_m = \arg min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m))$$

donde $\Theta_m = \{R_{jm}, \gamma_{jm}\}, j = 1,2, \dots, J_m$

Dadas las regiones R_{jm}, la obtención de los valores γ_{jm} se hace también mediante un proceso de minimización de pasos hacia delante:

$$\hat{\gamma}_{jm} = \arg min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L\big(y_i, f_{m-1}(x_i) + \gamma_{jm}\big)$$

Encontrar las distintas regiones en la que dividir el espacio es muy difícil, incluso más que en el caso de un árbol simple. Sin embargo, para ciertos casos, el problema se simplifica.

En el caso de que estemos ante una función de pérdida basada en el error cuadrático, la solución de $\widehat{\Theta}_m = \arg min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m))$ es, simplemente, el árbol de regresión que obtenga mejor predicción para los residuos actuales $y_i - f_{m-1}(x_i)$, y $\hat{\gamma}_{jm}$ viene dado por la media de los residuos de cada región obtenida.

Para un problema de clasificación con dos niveles o clases, utilizando una función de pérdida exponencial, este enfoque por pasos hacia delante da lugar al método AdaBoost utilizando árboles de clasificación, el cual vimos anteriormente. En particular, si los árboles $T(x; \Theta_m)$ están restringidos a ser árboles de clasificación ponderados, entonces la solución de $\widehat{\Theta}_m = \arg min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m))$ es el árbol que minimiza la tasa de error ponderado $\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq T(x_i, \Theta_m))$ con pesos $w_i^{(m)} = e^{-y_i f_{m-1}(x_i)}$, como mostramos ya en la sección……. Con una clasificación ponderada nos referimos a $\beta_m T(x; \Theta_m)$ con la restricción de que $\gamma_{jm} \in \{-1,1\}$. Sin esta restricción, la expresión

$$\widehat{\Theta}_m = \arg min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m))$$

quedaría de la siguiente manera:

$$\widehat{\Theta}_m = \arg min_{\Theta_m} \sum_{i=1}^{N} w_i^{(m)} e^{-y_i T(x_i, \Theta_m)}$$

Es sencillo implementar un algoritmo de partición recursiva usando esta función de pérdida exponencial ponderada como un criterio de división. Dado el R_{jm}, se puede mostrar (Ejercicio 10.7) que la solución a $\hat{\gamma}_{jm} = \arg min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma_{jm}\right)$ es la log-odds ponderada en cada región correspondiente

$$\hat{\gamma}_{jm} = \log \frac{\sum_{x_i \in R_{jm}} w_i^{(m)} I(y_i = 1)}{\sum_{x_i \in R_{jm}} w_i^{(m)} I(y_i = -1)}$$

Esto requiere un algoritmo especializado en el crecimiento de árboles; en la práctica, preferimos la aproximación presentada a continuación (Gradient Boosting) que usa un árbol de regresión de mínimos cuadrados ponderados.

El uso de criterios o funciones de pérdida como las basadas en el error absoluto o en el criterio de Huber en lugar de la función de pérdida basada en el error cuadrático para la regresión y la devianza en lugar de la función de pérdida exponencial para la clasificación servirán para robustecer los árboles en el modelo de Boosting. Sin embargo, estos criterios robustos no dan lugar a algoritmos simples y rápidos de Boosting.

Para los criterios de pérdida más generales, la solución a $\hat{\gamma}_{jm} = \arg min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma_{jm}\right)$, dada la R_{jm}, es generalmente sencilla ya que es una estimación simple de "ubicación". Para la función de pérdida basada en el valor absoluto, se trata solo de la mediana de los residuos en cada región.

Para los otros criterios existen algoritmos iterativos rápidos para resolver $\hat{\gamma}_{jm} = \arg min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma_{jm}\right)$, y generalmente sus aproximaciones de "un solo paso" más rápidas son adecuadas. El problema es la inducción de árboles, ya que no existen algoritmos simples y rápidos para resolver

$$\widehat{\Theta}_m = \arg min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m))$$

para estos criterios de pérdida más generales y, por tanto, las aproximaciones como

$$\widetilde{\Theta} = \arg min_{\Theta} \sum_{i=1}^{N} \tilde{L}(y_i, T(x_i, \Theta))$$

se vuelven esenciales.

m. Optimización numérica via "gradient boosting"

Los algoritmos de aproximación más rápidos para resolver

$$\widehat{\Theta}_m = \arg min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m))$$

para cualquier función de pérdida diferenciable se pueden derivar a la optimización numérica. La pérdida total es la suma de la función de pérdida aplicada a cada observación, es decir,

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$$

El objetivo principal es minimizar $L(f)$ con respecto a f tanto como sea posible, donde f está obligado a ser una suma de árboles

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_M)$$

Ignorando esta obligación, la minimización de $L(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$ puede ser vista como una optimización numérica

$$\hat{f} = \arg min_f L(f)$$

donde f viene dado por un vector N-dimensional ($f \in \mathbb{R}^N$) que contiene el resultado de aplicar f a cada una de las observaciones:

$$f = (f(x_1), f(x_2), \dots, f(x_N))$$

Resolvemos $\hat{f} = \arg min_f L(f)$ como una suma de componentes vectoriales

$$f_M = \sum_{m=0}^{M} h_m, \quad h_m \in \mathbb{R}^N$$

donde, inicialmente, $f_0 = h_0$ y luego, f_m se calcula teniendo en cuenta el valor de f_{m-1}, el cual es la suma de las actualizaciones inducidas previamente. Los métodos de optimización numérica se diferencian en la forma de calcular cada vector incremental h_m.

Descenso más rápido

Este método elige $h_m = -\rho_m g_m$ donde ρ_m es un escalar y $g_m \in \mathbb{R}^N$ es el gradiente de $L(f)$ evaluado en $f = f_{m-1}$. Es decir,

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i)}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$$

son las distintas componentes del gradiente g_m.

La longitud de paso es la solución a

$$\rho_m = \arg min_\rho \, L(f_{m-1} - \rho g_m)$$

Tras esto, se actualiza la solución a

$$f_m = f_{m-1} - \rho_m g_m$$

y se vuelve a repetir el proceso en la siguiente iteración. Este método destaca por ser donde $L(f)$ decrece más rápidamente debido a que se va optimizando en la dirección del gradiente, que es la de máximo descenso.

Gradient boosting

El algoritmo de Boosting por etapas hacia adelante es también una estrategia muy codiciosa. En cada paso, el árbol que tenemos como solución es el que más reduce $\widehat{\Theta}_m = \arg min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m))$, dado el modelo actual f_{m-1}. De esta forma, las predicciones dadas por $T(x_i, \Theta_m)$ son análogas a las componentes del gradiente con signo negativo $g_{im} = \left[\frac{\partial L(y_i, f(x_i)}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$. La principal diferencia entre ambas es que las componentes procedentes de los árboles $t_m = (T(x_1, \Theta_m), T(x_2, \Theta_m), \dots, T(x_N, \Theta_m))$ no son independientes. Están obligadas a ser las predicciones de un árbol de decisión de nodo J_m-terminal, mientras que el gradiente negativo es la dirección de máximo descenso sin restricciones.

La solución a $\hat{\gamma}_{jm} = \arg min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma_{jm}\right)$ en el método de pasos hacia adelante es análoga a la de $\rho_m = \arg min_\rho \, L(f_{m-1} - \rho g_m)$ en el caso del método de máximo descenso, con la diferencia de que en el primero se realiza una búsqueda separada en línea para aquellas componentes de t_m que corresponden a cada región terminal $\{T(x_i, \Theta_m)\}0_{x_i \in R_{jm}}$.

Si minimizar la función de pérdida en el conjunto de datos de entrenamiento $L(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$ fuese el único objetivo, el algoritmo de máximo descenso sería la mejor opción. El gradiente es fácil de calcular para cualquier función de perdida diferenciable, mientras que resolver $\widehat{\Theta}_m = \arg min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m))$ es difícil debido al ya mencionado criterio de robustez. Desafortunadamente, el gradiente solo está definido para las observaciones del conjunto de entrenamiento, siendo el principal y último objetivo generalizar la expresión $f_M(x)$ a los datos del conjunto test.

Una posible solución a este dilema presentado es inducir un árbol $T(x, \Theta_m)$ en la m-ésima iteración cuyas predicciones t_m estén tan cerca como sea posible del gradiente negativo. Esto nos conduce a

$$\widetilde{\Theta}_m = \arg min_{\Theta_m} \sum_{i=1}^{N} (-g_{im} - T(x_i, \Theta))^2$$

Explicándolo con mayor detalle, se trata de ajustar un árbol T a los valores del gradiente con signo negativo mediante mínimos cuadrados. Como vimos anteriormente, existen algoritmos rápidos para inducir árboles de decisión mediante mínimos cuadrados. Aunque las regiones $\tilde{R}_{jm}$que obtenemos mediante la solución a $\widetilde{\Theta}_m = \arg min_{\Theta_m} \sum_{i=1}^{N} (-g_{im} - T(x_i, \Theta))^2$ no son idénticas a las obtenidas R_{jm} resolviendo $\widehat{\Theta}_m = \arg min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m))$, sí podemos decir que son similares, ya que persiguen un mismo objetivo. En cualquier caso, el procedimiento de Boosting mediante pasos hacia adelante y la inducción de un árbol de decisión de manera descendente son procedimientos de aproximación. Tras construir el árbol mediante la expresión $\widetilde{\Theta}_m = \arg min_{\Theta_m} \sum_{i=1}^{N} (-g_{im} - T(x_i, \Theta))^2$, el valor que toman las observaciones que caen en una región viene dado por la expresión

$$\hat{\gamma}_{jm} = \arg min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma_{jm}\right)$$

Implementaciones del "gradient boosting"

1- REGRESION

1. Iniciamos con $f_0(x) = \sum_{i=1}^{N} L(y_i, \gamma)$

2. Desde $m = 1$ hasta $m = M$ calculamos

   a. Para $i = 1,2, \dots, N$

   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

   b. Ajustar un árbol de regresión a los objetivos r_{im} especificando las regiones terminales $R_{jm}, j = 1,2, \dots, J_m$

   c. Para cada región calcular:

   $$\gamma_{jm} = \arg min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

   d. Actualizar $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

3. Como salida obtendríamos $\hat{f}(x) = f_M(x)$

2- CLASIFICACIÓN

1. Iniciamos con $f_0(x) = \sum_{i=1}^{N} L(y_i, \gamma)$

2. Desde $k = 1$ hasta $k = K$ calculamos

    a. Desde $m = 1$ hasta $m = M$ calculamos

        i. Para $i = 1,2, \dots, N$

$$r_{ikm} = -\left[\frac{\partial L(y_i, f_{1m}(x_i), \dots, f_{1m}(x_i))}{\partial f_{km}(x_i)}\right]_{f=f_{m-1}}$$

        ii. Ajustar un árbol de regresión a los objetivos r_{ikm} especificando las regiones terminales $R_{jkm}, j = 1,2, \dots, J_m$

        iii. Para cada región calcular:

$$\gamma_{jkm} = \arg min_{\gamma} \sum_{x_i \in R_{jm}} L\left(y_i, f_{k,m-1}(x_i) + \gamma\right)$$

        iv. Actualizar $f_{km}(x) = f_{k,m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jkm} I(x \in R_{jkm})$

3. Como salida obtendríamos $\hat{f}(x) = f_M(x) = (f_{1M}(x), \dots, f_{KM}(x))$

n. Tamaño de los árboles en Boosting

Históricamente, el Boosting ha sido considerado como una técnica de combinación de modelos (en este caso árboles). El algoritmo de construcción de un árbol ha sido visto como una forma primitiva de producir modelos para que éstos sean combinados por el procedimiento de Boosting. En este escenario, el tamaño óptimo de cada árbol se estima mediante su construcción. Se construye un árbol muy extenso y luego se poda al número óptimo de nodos. Este enfoque supone implícitamente que cada árbol es el último en la expansión

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_M)$$

Excepto quizás para el último árbol, esta es claramente una suposición muy pobre. El resultado es que los árboles tienden a ser demasiado grandes, especialmente durante las primeras iteraciones. Esto degrada sustancialmente el rendimiento y aumenta el cálculo o computación.

La estrategia más simple para evitar este problema es restringir todos los árboles a tener el mismo número de nodos, es decir, $J_m = J \; \forall m$. De este modo, como en cada etapa hay que inducir un árbol de regresión de tamaño J, éste se convierte en un parámetro de todo el procedimiento Boosting, que se ajustará para maximizar el rendimiento estimado de los datos disponibles.

Podemos obtener una idea de los valores de J más útiles considerando las propiedades de la función objetivo

$$\eta = \arg min_{\mathrm{f}} E_{XY} L(Y, f(X))$$

El valor esperado es sobre la distribución conjunta de la población (X, Y).

La función objetivo $\eta(x)$ es la que tiene un riesgo de predicción mínimo en los datos del conjunto test, por tanto, es la que vamos a intentar aproximar. Una propiedad relevante de $\eta(X)$ es el grado en que las distintas variables predictoras $X^T = (X_1, X_2, \dots, X_p)$ interactúan entre sí. Esto se obtiene mediante por su expansión ANOVA

$$\eta(X) = \sum_j \eta_j(X_j) + \sum_{jk} \eta_{jk}(X_j, X_k) + \sum_{jkl} \eta_{jkl}(X_j, X_k, X_l) + \cdots$$

El primer sumando es sobre funciones de una única variable predictora X_j. Las funciones $\eta_j(X_j)$ son aquellas que, conjuntamente, aproximan mejor $\eta(X)$ bajo el criterio o función de pérdida que está siendo utilizado. Cada función $\eta_j(X_j)$ es llamada "efecto principal" de X_j. El segundo sumando es sobre aquellas funciones con dos variables predictoras que, añadidas a los efectos principales, ajustan mejor $\eta(X)$. Estas son llamadas interacciones de segundo orden de cada respectivo par de variables (X_j, X_k). El tercer sumando representa las interacciones de tercer orden y así sucesivamente. Debido a muchos problemas encontrados en la práctica, los efectos de interacción de bajo orden tienden a dominar. Cuando este es el caso, los modelos que producen fuertes efectos de interacción de orden superior, como los grandes árboles de decisión, sufren de precisión.

El nivel de interacción de las aproximaciones basadas en árboles está limitado por el tamaño del árbol J, ya que no hay efectos de interacción de un nivel mayor que $J - 1$. Esto sugiere que el

valor elegido para J debe reflejar el nivel de interacciones dominantes de $\eta(X)$. Por supuesto, esto es generalmente desconocido, pero en la mayoría de las situaciones tenderá a ser bajo.

o. Regularización

Además del tamaño de los árboles que constituyen un procedimiento de Boosting, otro parámetro importante del "gradient Boosting" es el número de iteraciones M. Normalmente, en cada iteración se reduce el error de entrenamiento, de modo que para un M suficientemente grande este error puede hacerse muy pequeño. Sin embargo, ajustar los datos de entrenamiento de una manera tan precisa puede llevarnos al sobreajuste, lo cual no es bueno para futuras predicciones. Por lo tanto, existe un número óptimo M^* que minimiza el error de predicciones futuras. Una forma conveniente de estimar M^*es calcular el riesgo o error de predicción en función de M en una muestra de validación. El valor de M que minimiza este riesgo se toma como una estimación de M^*.

Shrinkage

Controlar el valor de M no es la única estrategia de regularización posible. También se pueden emplear técnicas de Shrinkage (contracción) que, en el contexto del boosting, consiste en ponderar la contribución de cada árbol por un factor $0 < v < 1$ cuando se agrega a la aproximación actual. Esto anterior supondría cambiar en el algoritmo de Gradient Boosting para regresión la expresión que aparece en la línea d) por

$$f_m(x) = f_{m-1}(x) + v \sum_{j=1}^{J_m} \gamma_{jm} I\left(x \in R_{jm}\right)$$

Se puede considerar que el parámetro v puede controlar la velocidad de aprendizaje del procedimiento de Boosting. Los valores más pequeños de v (más contracción) generan un mayor riesgo o error de entrenamiento para el mismo número de iteraciones M. Por lo tanto, tanto v como M controlan el riesgo de predicción en los datos de entrenamiento. Sin embargo, estos parámetros no funcionan de manera independiente. Los valores más pequeños de v conducen a valores más grandes de M para compensar y, de esta forma, mantener el error o riesgo de entrenamiento.

Empíricamente se ha demostrado (Friedman, 2001) que los valores más pequeños de v conducen a un error de test menor. De hecho, la mejor estrategia parece ser establecer que v sea muy pequeña ($v < 0.1$) y luego elegir M. Esto produce muy buenas mejoras para la regresión y para la estimación de la probabilidad. El precio que hay pagar por estas mejoras es

computacional ya que valores menores de v dan lugar a valores mayores de M, y el cálculo es proporcional a este último.

Submuestreo (stochastic gradient boosting)

Con el "stochastic gradient boosting" (Friedman, 1999), en cada iteración se muestrea una fracción η de las observaciones de entrenamiento (sin reemplazamiento), y se construye el siguiente árbol usando esa submuestra. El resto del algoritmo es idéntico. Un valor típico para η puede ser $1/2$, aunque para N grande, η puede ser sustancialmente menor que $1/2$.

El submuestreo no solo reduce el tiempo de cálculo, sino que en muchos casos produce un modelo mucho más preciso debido a que estamos eliminando riesgo de sobreajuste.

p. Distintos métodos Boosting

DOS CLASES

Algoritmo Real AdaBoost

Suponiendo que las clases son +1 y -1, el clasificador base devuelve una estimación de la probabilidad de pertenencia a la clase +1.

1- En principio, los pesos de las observaciones tendrán el mismo valor:

$$w_i = \frac{1}{n}, i = 1, \dots, n$$

2- Desde $m = 1$ hasta $m = M$

   a. Entrenar el clasificador teniendo en cuenta los w_i para obtener una estimación de la probabilidad de pertenencia a la clase +1.

   $$p_m(x) = \hat{p}_w(y = +1|x) \in [0,1]$$

   b. Hacer $f_m(x) = \frac{1}{2}\log\left(\frac{p_m(x)}{1-p_m(x)}\right) \in \mathbb{R}$

   c. Actualizar los pesos de las observaciones

   $$w_i = w_i e^{-y_i f_m(x_i)}$$

   d. Escalar los pesos para que sumen uno:

   $$w_i = \frac{w_i}{\sum_{j=1}^{n} w_j}$$

3- El clasificador final viene dado por:

$$C(x) = sign\left[\sum_{m=1}^{M} f_m(x)\right]$$

Algoritmo Gentle AdaBoost

1- Inicializamos con:

$$w_i = \frac{1}{n}, i = 1, \dots, n \quad \text{y} \quad F(x) = 0$$

2- Desde $m = 1$ hasta $m = M$

a. Determinar la función $f_m(x)$ mediante regresión de mínimos cuadrados ponderada de y_i respecto a x_i con pesos w_i.

b. Actualizar $F(x)$

$$F(x) = F(x) + f_m(x)$$

c. Actualizar los pesos

$$w_i = w_i e^{-y_i f_m(x_i)}$$

d. Escalar los pesos para que sumen uno:

$$w_i = \frac{w_i}{\sum_{j=1}^{n} w_j}$$

3- El clasificador final viene dado por:

$$C(x) = sign\left[\sum_{m=1}^{M} f_m(x)\right] = sign\,[F(x)]$$

Algoritmo LogitBoost

1- Inicializamos con:

$$w_i = \frac{1}{n}, i = 1, \dots, n, \quad F(x) = 0 \text{ y } p(x) = \frac{1}{2}$$

2- Desde $m = 1$ hasta $m = M$

a. Para $i = 1, \dots, n$ calcular:

$$z_i = \frac{y_i^* - p(x_i)}{p(x_i)\big(1 - p(x_i)\big)} \; con \; y_i^* = \frac{y_i + 1}{2}$$

$$w_i = p(x_i)\big(1 - p(x_i)\big)$$

b. Determinar la función $f_m(x)$ mediante regresión de mínimos cuadrados ponderada de y_i respecto a x_i con pesos w_i.

c. Actualizar:

$$F(x) = F(x) + \frac{1}{2} f_m(x)$$

$$p(x) = \frac{e^{F(x)}}{e^{F(x)} + e^{-F(x)}}$$

3- El clasificador final viene dado por:

$$C(x) = sign\left[\sum_{m=1}^{M} f_m(x)\right]$$

MÁS DE DOS CLASES

Algoritmo AdaBoost.M1

1- En principio, los pesos de las observaciones tendrán el mismo valor:

$$w_i = \frac{1}{n}, i = 1, \dots, n$$

2- Desde $m = 1$ hasta $m = M$

a. Obtener un clasificador que minimice el error ponderado $\mathcal{E}_m$.

$$\mathcal{E}_m = \sum_{y_i \neq C_m(x_i)} w_i$$

b. Hacer $\alpha_m(x) = \frac{1}{2}\log\left(\frac{1-\mathcal{E}_m}{\mathcal{E}_m}\right)$

c. Actualizar los pesos de las observaciones

$$w_i = w_i e^{\alpha_m(x) I(y_i \neq C_m(x_i))}$$

d. Escalar los pesos para que sumen uno:

$$w_i = \frac{w_i}{\sum_{j=1}^{n} w_j}$$

3- El clasificador final viene dado por:

$$C(x) = \arg max_{j \in Y} \sum_{m=1}^{M} \alpha_m I(C_m(x) = j) \ \ con\ Y = \{1, \dots, K\}\ clases$$

Algoritmo SAMME

1- En principio, los pesos de las observaciones tendrán el mismo valor:

$$w_i = \frac{1}{n}, i = 1, \dots, n$$

2- Desde $m = 1$ hasta $m = M$

e. Obtener un clasificador que minimice el error ponderado $\mathcal{E}_m$.

$$\mathcal{E}_m = \sum_{y_i \neq C_m(x_i)} w_i$$

f. Hacer $\alpha_m(x) = \frac{1}{2}\log\left(\frac{1-\mathcal{E}_m}{\mathcal{E}_m}\right) + \ln(K-1)\ \ con\ K\ nº\ de\ clases$

g. Actualizar los pesos de las observaciones

$$w_i = w_i e^{\alpha_m(x) I(y_i \neq C_m(x_i))}$$

h. Escalar los pesos para que sumen uno:

$$w_i = \frac{w_i}{\sum_{j=1}^{n} w_j}$$

3- El clasificador final viene dado por:

$$C(x) = \arg max_{j \in Y} \sum_{m=1}^{M} \alpha_m I(C_m(x) = j) \ \ con\ Y = \{1, \dots, K\}\ clases$$

Algoritmo AdaBoost.MH

1- Expandir las n observaciones en $n \times K$ pares de la forma:

$$((x_i, 1), y_{i1}), ((x_i, 2), y_{i2}), \dots, ((x_i, K), y_{iK}) \quad i = 1, \dots, n$$

Con $y_{ij} = \begin{cases} 1 & si\ x_i\ pertenece\ a\ la\ clase\ j \\ -1 & en\ caso\ contrario \end{cases}$

2- Aplicar Real AdaBoost a la muestra aumentada para obtener una función

$$F: \mathbb{R}^d \times Y \to \mathbb{R}\ \ dada\ por\ F(x, j) = \sum_{m=1}^{M} f_m(x, j)$$

3- Clasificador final:

$$C(x) = \arg max_{j \in Y}\ F(x, j)$$

Algoritmo LogitBoost

1- Inicializamos con:

$$w_{ij} = \frac{1}{n}, i = 1, \dots, n, \quad F_j(x) = 0 \text{ y } p_j(x) = \frac{1}{2} \quad j = 1, \dots, K$$

2- Desde $m = 1$ hasta $m = M$

a. Desde $j = 1$ hasta $j = K$

ii. Para $i = 1, \dots, n$ calcular:

$$z_{ij} = \frac{y_{ij}^* - p_j(x_i)}{p_j(x_i)\left(1 - p_j(x_i)\right)}\ con\ y_{ij}^* = \frac{y_{ij} + 1}{2}$$

$$w_i = p_j(x_i)\left(1 - p_j(x_i)\right)$$

iii. Determinar la función $f_{mj}(x)$ mediante regresión de mínimos cuadrados ponderada de y_{ij} respecto a x_i con pesos w_{ij}.

iv. Calcular y Actualizar:

$$f_{mj}(x) = \frac{K - 1}{K}\left(f_{mj}(x) - \frac{1}{K}\sum_{l=1}^{K} f_{ml}(x)\right)$$

$$F_j(x) = F_j(x) + \frac{1}{2} f_{mj}(x)$$

$$p(x) = \frac{e^{F_j(x)}}{e^{F_j(x)} + e^{-F_j(x)}}$$

3- El clasificador final viene dado por:

$$C(x) = \arg max_{j \in Y}\ F_j(x)$$

7. APLICACIÓN CON R

a. Modelos “Tipificación”

Ahora vamos a mostrar un ejemplo práctico de todo o casi todo lo que hemos ido comentando en este documento.

La idea u objetivo de esta sección va a ser clasificar una serie de incidencias de una de las mayores empresas de telecomunicaciones de este país en función del detalle (variables resumen, detalle y resolución) que tengamos de cada una de ellas.

Contaremos con un total de 1903 incidencias, las cuales dividiremos en conjunto de entrenamiento (1461 observaciones) y conjunto test (487 observaciones). En cuanto a tipificaciones (o clases) distintas hay un total de 21 pero nos quedaremos con las que tengan más de 25 ocurrencias en todo el conjunto de incidencias, ya que hay algunas muy inusuales y, por tanto, muy difíciles de predecir. Las 12 tipificaciones con más de 25 ocurrencias son las siguientes:

```
 [1] "Incidencia sin actuación por causa origen no perteneciente a Middleware Gfi"
 [2] "Incidencia con actuación a petición por causa origen no perteneciente a Middleware Gfi"
 [3] "Incidencia por llenado de FS perteneciente a middleware"
 [4] "Incidencia en el middleware por estado anómalo de la instancia/proceso/servicio"
 [5] "Incidencia por problema de configuración previo"
 [6] "Incidencia no reproducida/error puntual"
 [7] "Incidencia provocada cambio en curso"
 [8] "Incidencia mal tipificada por corresponderse a una petición"
 [9] "Incidencia en el middleware por caída de la instancia/proceso/servicio"
[10] "Incidencia relacionada con bug de software"
[11] "Incidencia en el middelware por FullGC"
[12] "Incidencia provocada por falso positivo en la monitorización"
```

Debido a que el campo “Detalle” (unión de los tres campos textuales) es un campo tipo texto, realizaremos un análisis de texto para convertir ese campo textual en muchos campos numéricos. En primer lugar, crearemos un vector con las distintas palabras que contenga el campo “Detalle”. Tras eliminar de este vector palabras irrelevantes con la librería “stopwords” y palabras compuestas por caracteres extraños, lematizamos todas las palabras y eliminamos repetidas. Finalmente, nos quedamos con las que tengan más de 20 ocurrencias en el campo “detalle” entre todas las observaciones. En total nos quedamos con un diccionario de 1237 palabras.

A continuación, creamos la matriz o dataframe a la que le vamos a aplicar los algoritmos de AdaBoostM1, Bagging, Stochastic Gradient Boosting y Random Forest. Esta matriz tendrá tantas

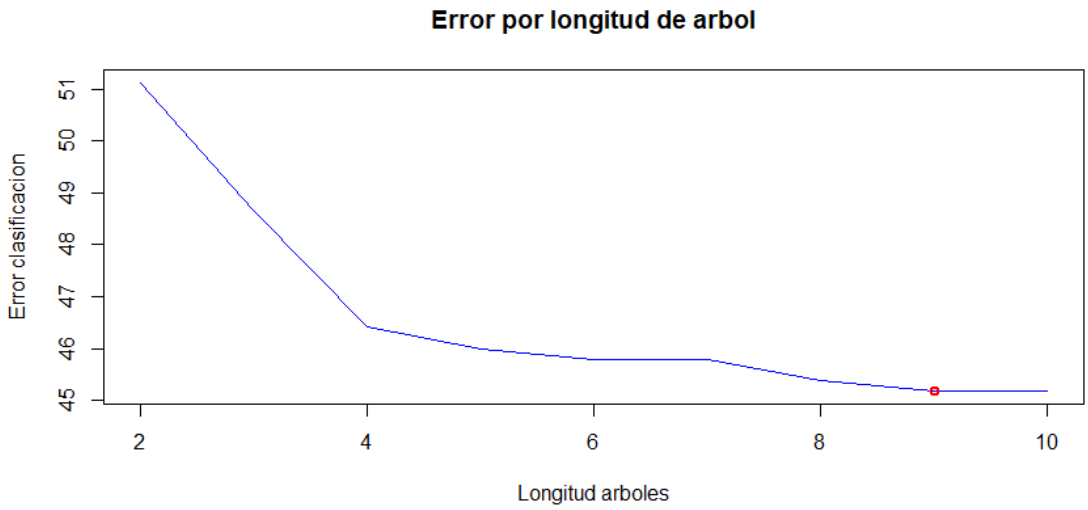
filas como observaciones haya en el conjunto de entrenamiento y cada palabra de nuestro diccionario será una columna. En la columna correspondiente a la palabra “j” aparecerá un 1 si la palabra “j” aparece en el detalle de la observación. En caso contrario aparecerá un 0. Es decir, vamos a tener una matriz de dimensiones 1461 x 1237 rellena de 0`s y 1`s, a la que se le añadirá la columna “tipificación”.

Estudio mediante método AdaBoostM1

Para llevar a cabo este estudio usaremos la librería “adabag”.

Construiremos 9 modelos de 50 árboles cada uno en función de la longitud de cada uno de los árboles. Es decir, la longitud de los árboles del primero estará restringida a 2 y la longitud de los árboles del último estará restringida a 10.

Para cada uno de los modelos realizaremos las predicciones en el conjunto test y obtendremos el error de clasificación de cada uno de ellos. La gráfica de los errores en función de la longitud de los árboles quedaría de la siguiente forma:



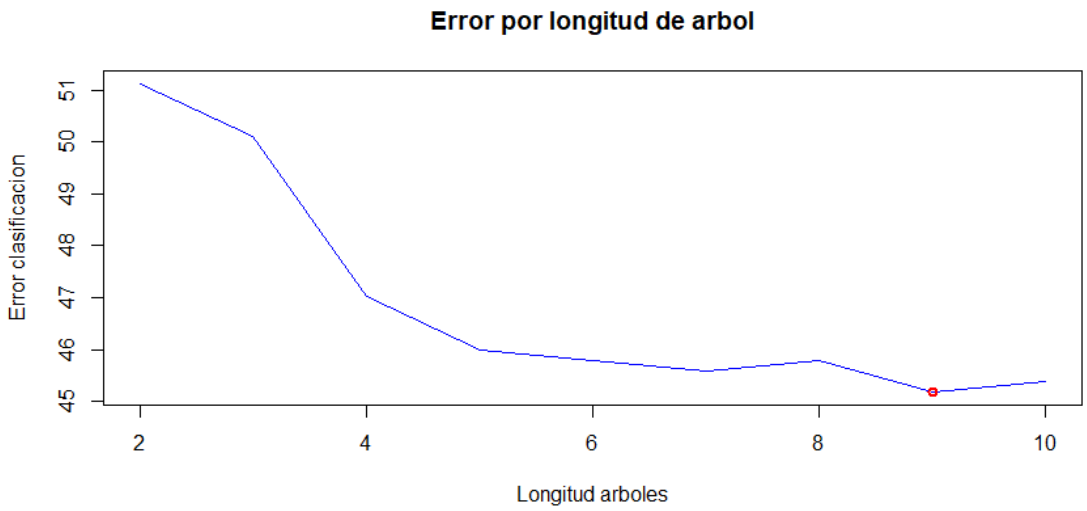
Como se puede ver en el gráfico, el modelo con menor error de predicción es el que usa árboles de longitud 9. Por tanto, nuestro modelo AdaBoostM1 definitivo usará árboles con esa longitud, obteniendo un error de clasificación del 45,38%.

Estudio mediante método Bagging

Para llevar a cabo este estudio también usaremos la librería “adabag”.

Al igual que en el método anterior, construiremos 9 modelos de 50 árboles cada uno en función de la longitud de cada uno de los árboles y, para cada uno de los modelos realizaremos las predicciones en el conjunto test y obtendremos el error de clasificación de cada uno de ellos.

De igual forma, la gráfica de los errores en función de la longitud de los árboles quedaría de la siguiente manera:

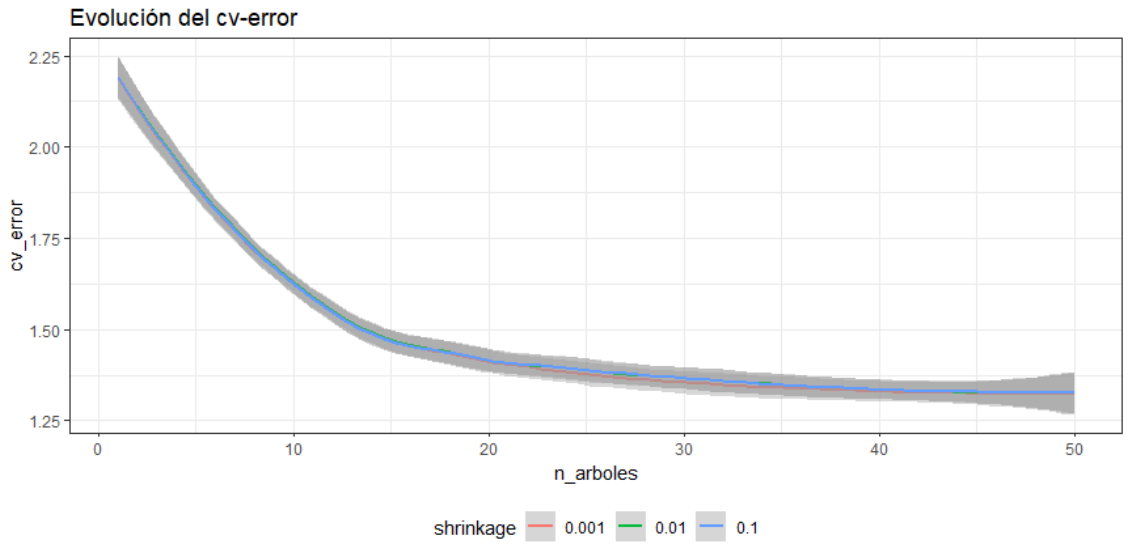


Como la gráfica muestra, el modelo con menor error de predicción es el que usa árboles de longitud 9, al igual que pasaba en el método AdaBoostM1. Por tanto, nuestro modelo Bagging definitivo usará árboles con esa longitud, obteniendo un error de clasificación del 45,38%.

Estudio mediante método Stochastic Gradient Boosting

La principal diferencia entre el uso del algoritmo de Stochastic Gradient Boosting y los demás es que, a la hora de construir cada árbol, utilizaremos el 50% de las observaciones de la muestra de entrenamiento. Además, también elegiremos el mejor modelo en función de varios parámetros. Para llevar a cabo este estudio utilizaremos la librería “gbm”.

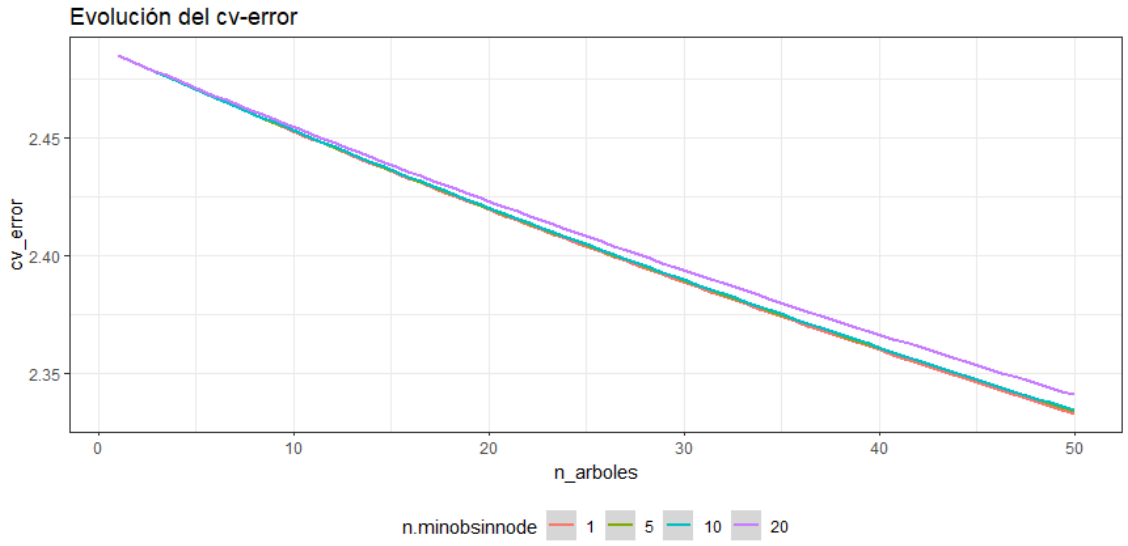
En primer lugar, ajustaremos el parámetro “shrinkage” considerando procedimientos Boosting con un total de 50 iteraciones (árboles). Mediante validación cruzada obtendremos una estimación del error de test en función del número de árboles usados y el valor del parámetro de contracción (shrinkage). Como vamos a ver a continuación en la gráfica, hemos considerado 3 valores de “shrinkage”:



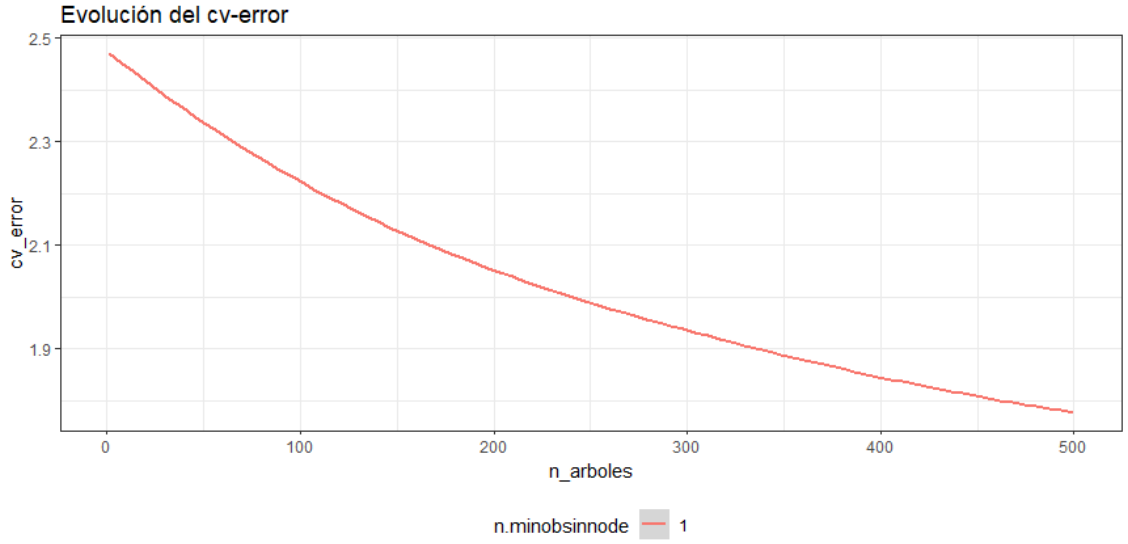
Como podemos observar, obtenemos prácticamente el mismo error para los 3 valores del parámetro. Aunque siendo precisos, nos quedaremos con el valor 0.001, ya que tiene el menor error.

	cv_error	n_arboles	shrinkage
50	1.320256	50	0.001
100	1.327172	50	0.010
150	1.324312	50	0.100

Ahora vamos a comparar la estimación del error de test en función de si el número mínimo de observaciones en un nodo es 1,5,10 ó 20. Obviamente, en estos procedimientos ya estamos usando el valor de shrinkage=0.001. La gráfica que obtenemos ahora es la siguiente:



Aquí podemos ver que la línea roja es ligeramente inferior a las demás. Por tanto, el parámetro de número mínimo de observaciones en un nodo vendrá dado por el número 1. Además, como vemos que el error tiende a seguir bajando ampliaremos el número de árboles a 500, donde vemos que se estabiliza un poco.

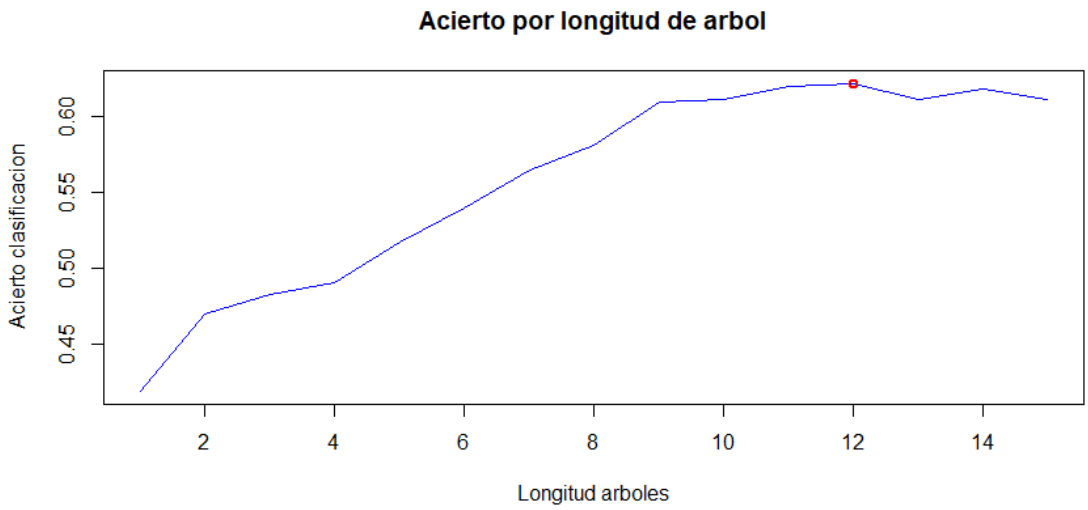


Tras ejecutar el algoritmo final obtenemos una tasa de fallo del 43.33%

Estudio mediante método Random Forest

Para aplicar Random Forest utilizaremos la librería “randomForest”.

Consideraremos un total de 15 modelos Random Forest, los cuales vendrán definidos en función del número de nodos terminales (desde 2^1 a 2^{15}). Calcularemos las predicciones de cada uno de ellos y la tasa de acierto. Vemos ahora una gráfica que nos muestra la tasa de acierto en función del número de nodos terminales:



La gráfica nos muestra como el modelo con mayor acierto en la predicción es el que tiene 2^{12} nodos terminales, el cual nos da una tasa de acierto de 61.40%

Comparación de modelos

Tras la ejecución de los distintos modelos, mostramos una tabla comparativa con el error que hemos obtenido en cada uno de ellos:

error_adaboostM1	error_bagging	error_gbm	error_rf
0.4537988	0.4537988	0.4332649	0.386037

Observamos que el método que nos da un mejor resultado es el Random Forest con un error de clasificación de un 0.386.

b. Modelos Carseats

Vamos a realizar ahora otro ejemplo con un conjunto de datos donde, a diferencia de nuestro primer caso, nuestra variable a predecir consta de dos clases. Utilizaremos el conjunto de datos "Carseats" de la librería ISLR, que contiene datos sobre ventas de asientos infantiles para automóviles en 400 tiendas diferentes. Consta de 400 observaciones y de 12 variables, que son las siguientes:

```
"Sales"       "CompPrice"   "Income"      "Advertising" "Population"
"Price"       "ShelveLoc"   "Age"         "Education"   "Urban"
"US"          "High"
```

La variable objetivo será High, la cual está formada en función de la variable Sales (unidades vendidas). Si Sales es mayor que 8 la variable High tomará el valor "Yes" y, en caso contrario, tomará el valor "No". Obviamente, a la hora de realizar los modelos, no tendremos en cuenta la variable Sales.

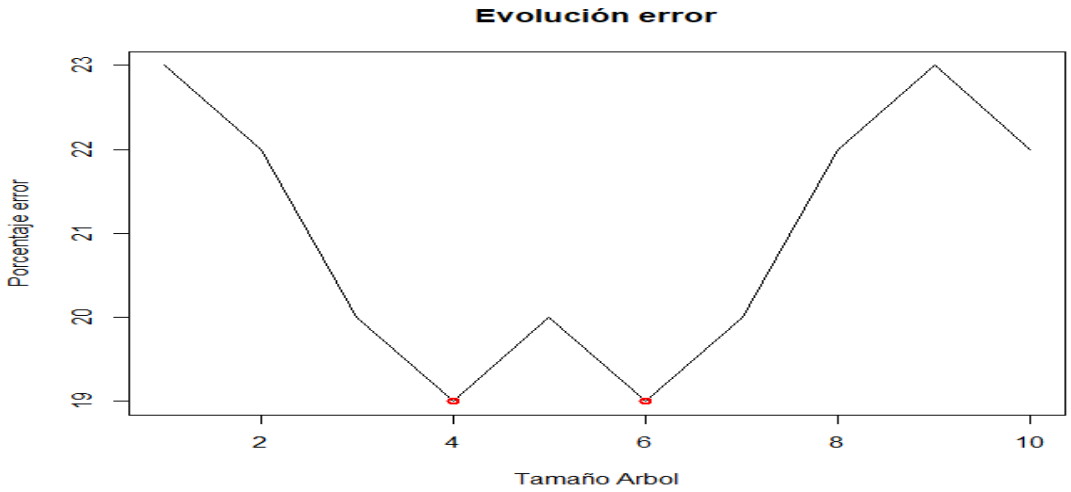
Dividiremos el conjunto de datos en conjunto de datos de entrenamiento (300 observaciones) y conjunto de datos test (100 observaciones).

Estudio mediante método AdaBoostM1

Para llevar a cabo este estudio usaremos la librería "adabag".

En este caso, construiremos 10 modelos de 50 árboles cada uno en función de la longitud de cada uno de los árboles.

Para cada uno de los modelos realizaremos las predicciones en el conjunto test y obtendremos el error de clasificación de cada uno de ellos. La gráfica del error de clasificación obtenido para cada modelo en función del tamaño es la siguiente:

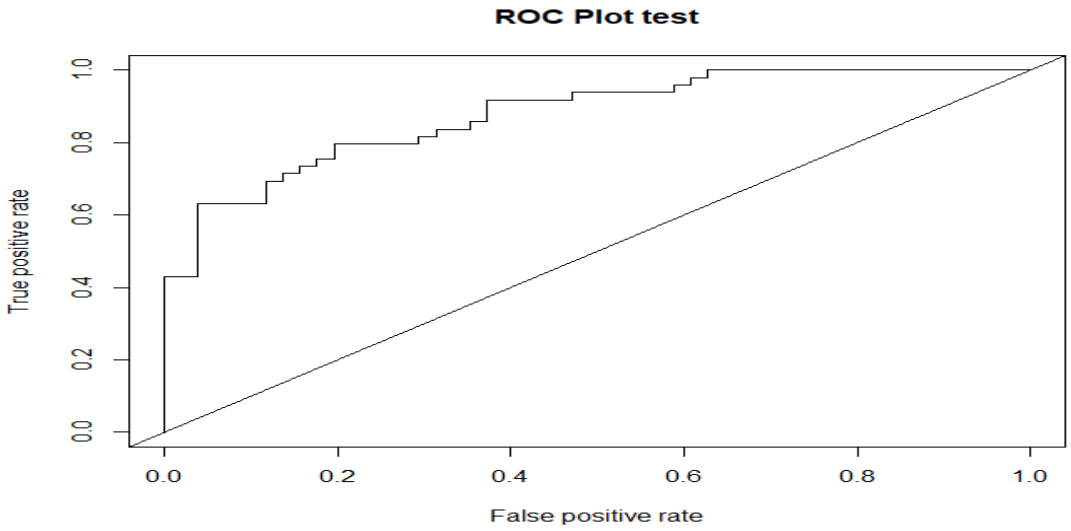


Vemos que obtenemos los errores más bajos en cuando el tamaño es 4 o es 6. Por tanto, nos quedamos con el valor 4, que es más bajo. Para el modelo elegido las matrices de confusión y la curva ROC son las siguientes:

- Matriz confusión:

	Observed Class	
Predicted Class	No	Yes
No	45	15
Yes	6	34

- Curva ROC:



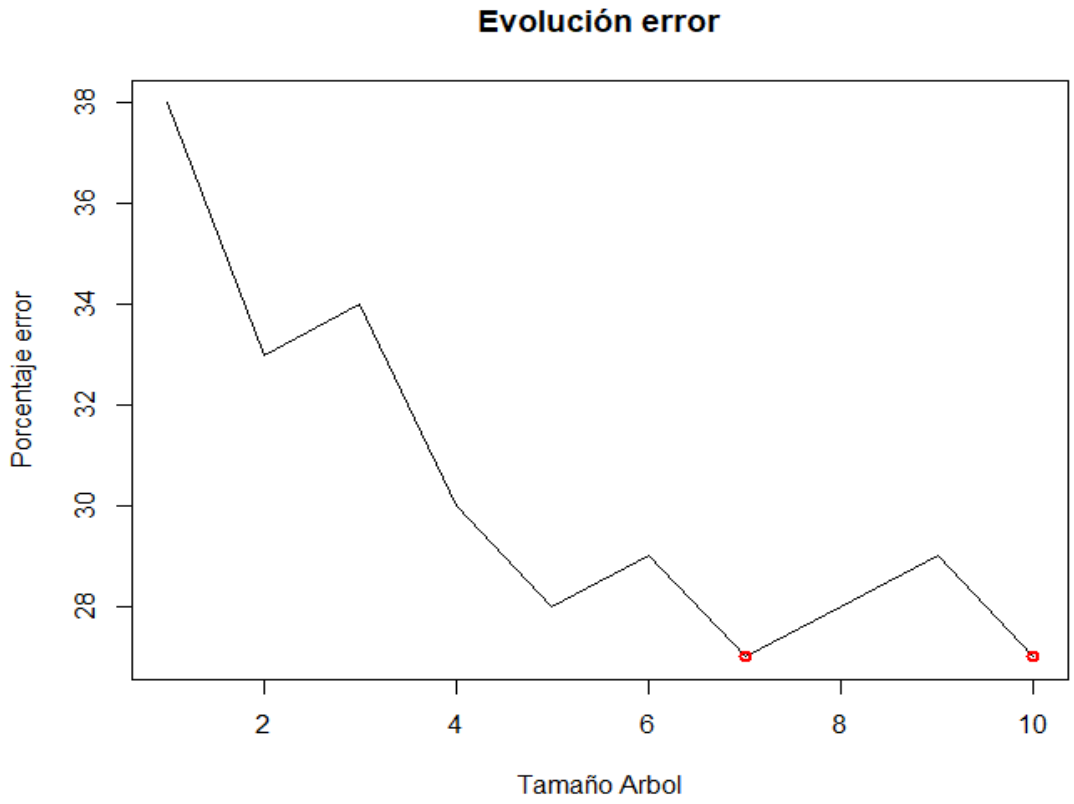
El error obtenido con este modelo es del 21%.

Estudio mediante método Bagging

Para llevar a cabo este estudio también usaremos la librería “adabag”.

Al igual que en el modelo anterior, construiremos 10 modelos de 50 árboles cada uno en función de la longitud de cada uno de los árboles y, para cada uno de los modelos realizaremos las predicciones en el conjunto test y obtendremos el error de clasificación de cada uno de ellos.

La gráfica de los errores en función de la longitud de los árboles quedaría de la siguiente forma:

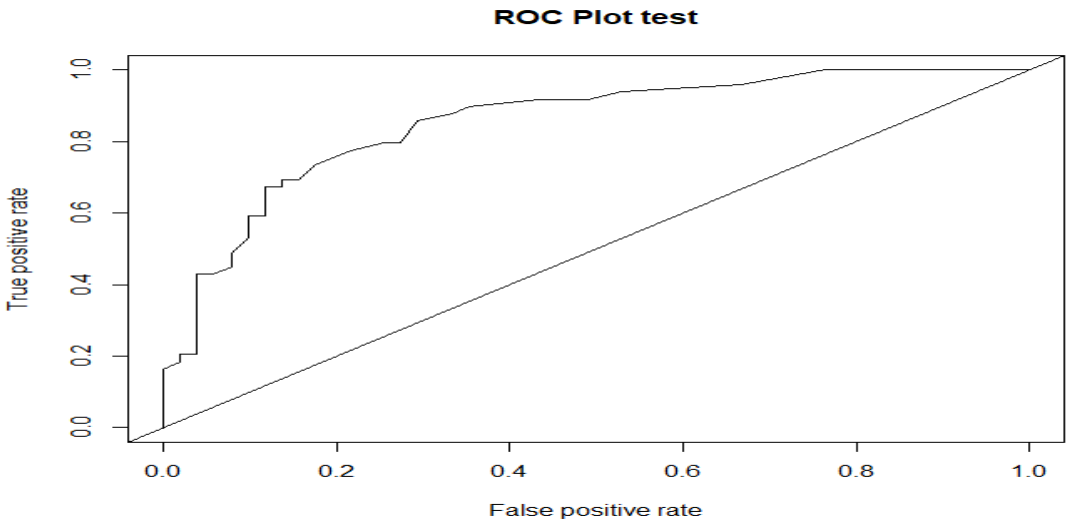


Elegimos el modelo con longitud 7 para realizar las predicciones, ya que es la mínima longitud de árbol con el menor error. La matriz de confusión y la curva ROC son las siguientes:

- Matriz confusión:

```
                Observed Class
redicted Class  No Yes
           No   46   20
           Yes   5   29
```

- Curva ROC:

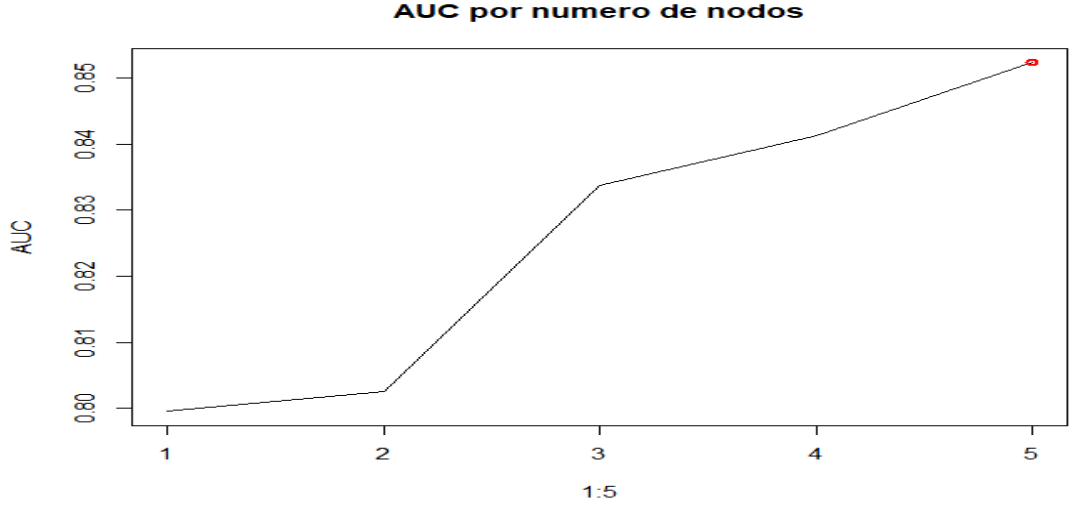


El error obtenido con este modelo es del 25%.

Estudio mediante método Random Forest

Para aplicar Random Forest utilizaremos la librería "randomForest".

En este ejemplo, consideraremos un total de 5 modelos Random Forest con 500 árboles cada uno, los cuales vendrán definidos en función del número de nodos terminales (desde 2^1 a 2^5). Calcularemos las predicciones de cada uno de ellos y la tasa de acierto. Mostramos ahora una gráfica que nos muestra el área bajo la curva ROC en función del número de nodos terminales:

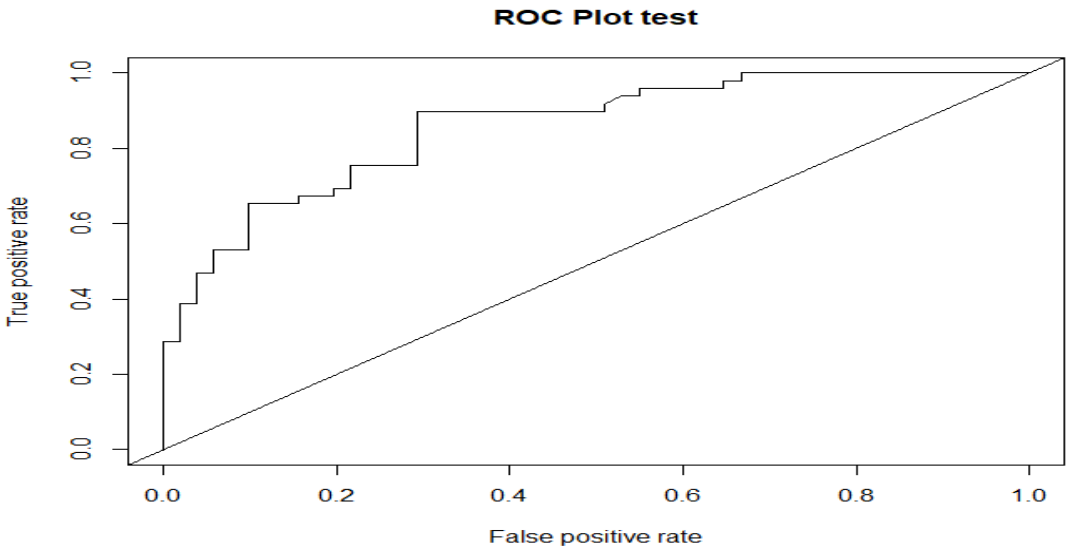


Vemos como el modelo con 5 nodos terminales tiene el área bajo la curva más alta. Por tanto, elegiremos el modelo que cuenta con 5 nodos terminales en cada árbol, con el que tenemos el 74% de acierto.

- Matriz confusión:

```
pred.rf No Yes
   No   46   21
   Yes   5   28
```

- Curva ROC:



Comparación de modelos

La tabla con los errores de los distintos modelos es la siguiente:

```
error_boost error_bag error_rf
       0.21      0.25     0.26
```

Por tanto, podemos deducir que el método más acertado es el AdaBoostM1 de la librería Adabag.

8. ANEXOS

- Script Datos Tipificaciones

PREPROCESAMIENTO

```
#TRABAJO FIN DE GRADO

#TECNICAS BOOSTING

#MANUEL PEREZ GARCIA

#DOBLE GRADO EN MATEMATICAS Y ESTADISTICA

#DEPARTAMENTO DE ESTADISTICA E INVESTIGACION OPERATIVA


####################################

#PREPROCESAMIENTO DE LOS DATOS

####################################


# LIBRERIAS QUE VAMOS A UTILIZAR

if (!"DBI" %in% rownames(installed.packages())) install.packages("DBI")

library(DBI)

if (!"RMySQL" %in% rownames(installed.packages())) install.packages("RMySQL")

library(RMySQL)

if (!"stringi" %in% rownames(installed.packages())) install.packages("stringi")

library(stringi)

if (!"SnowballC" %in% rownames(installed.packages())) install.packages("SnowballC")

library(SnowballC)

if (!"stopwords" %in% rownames(installed.packages())) install.packages("stopwords")

library(stopwords)


# Abrimos conexion con MySQL

# conexion =
dbConnect(dbDriver("MySQL"),host="194.140.7.161",dbname="genesis",user="root",passwor
d="genesis");

# consulta<-"SELECT

# *
```

```
# FROM

# dwh_ct_top5_inc

# where

# op_tiene_auditoria=1

# and Grupo_Servicio in ('Middleware','Procesos y Productos','Herramientas ciclo vida
SW','Herramientas DEVOPS')"

# tipificaciones= dbGetQuery(conexion,statement=consulta)

#

#

# # Cerramos conexion con MySQL

# dbDisconnect(conexion)



#Leemos fichero csv con las incidencias

tipificaciones<-read.csv2(file = "tipificaciones.csv",sep = ";",header = T,

                          stringsAsFactors = F)


#Nos quedamos solo con las incidencias tipificadas y con las variables tipo texto (ademas de la
variable Tipificacion)

tipificaciones<-tipificaciones[tipificaciones$op_tiene_auditoria==1,]

campos<-colnames(tipificaciones)

campos

campos<-campos[25:28]

tipificaciones_resumen<-tipificaciones[,campos]


#Guardamos el numero de campos

m<-length(campos)


#En caso de leer los datos desde BBDD hay que ejecutar lo siguiente por temas de formato

# for (i in 1:m){

#   Encoding(tipificaciones_resumen[,i]) = "UTF-8"

# }
```

```
#Vemos todos los tipos de tipificacion que tenemos

unique(tipificaciones_resumen$Tipificacion)



#Normalizamos los tipos y nos quedamos solo con los que tienen más de 25 apariciones

tipificaciones_resumen[tipificaciones_resumen$Tipificacion=="\nIncidencia en el middleware por estado anómalo de la instancia/proceso/servicio",]$Tipificacion="Incidencia en el middleware por estado anómalo de la instancia/proceso/servicio"

unique(tipificaciones_resumen$Tipificacion)

tipificaciones_resumen<-tipificaciones_resumen[(tipificaciones_resumen$Tipificacion!="Prueba")&(tipificaciones_resumen$Tipificacion!="prueba"),]

unique(tipificaciones_resumen$Tipificacion)

validas<-names(which(table(tipificaciones_resumen$Tipificacion)>25))

tipificaciones_resumen<-tipificaciones_resumen[tipificaciones_resumen$Tipificacion%in%validas,]

unique(tipificaciones_resumen$Tipificacion)



#Guardamos el numero de registros

n<-nrow(tipificaciones_resumen)



#Creamos un vector de palabras irrelevantes que tenemos que eliminar de nuestro posterior diccionario

elimina_ingles<-tolower(c(stopwords(language = "en"),stopwords("SMART")))

elimina_español<-tolower(stopwords(language = "es"))

elimina_particular<-tolower(c('http:','/','\\','hola','días','dias','tardes','buenos','buenas',

                              'favor','muchas','gracias','fecha','hora','unas',

                              'mañana','tarde','noche',

                              'lunes','martes','miercoles','miércoles','jueves','viernes',
```

```
                    'sábado','sabado','domingo'))
elimina<-c(elimina_ingles,elimina_español,elimina_particular)


#Unimos los campos textuales de Detalle y Resolucion que son los que mas informacion tienen acerca de las incidencias
tipificaciones_resumen$texto=paste(tipificaciones_resumen$Detalle,tipificaciones_resumen$Resolucion)
texto=tipificaciones_resumen$texto

#Convertimos el campo texto en un vector con las palabras que lo componen para cada incidencia (registro)
texto_split = strsplit(texto, split=" ")


#Funcion de limpieza de las palabras que contiene el campo texto
limpio<-function (texto_columnas){
 texto_columnas<-tolower(texto_columnas)
 texto_columnas<-unique(texto_columnas)
 texto_columnas<-gsub("[[:punct:]]","",texto_columnas)
 texto_columnas<-gsub("[[:digit:]]","",texto_columnas)
 texto_columnas<-gsub(" ","",texto_columnas)
 texto_columnas<-gsub("-","",texto_columnas)
 texto_columnas<-gsub("/","",texto_columnas)
 texto_columnas<-gsub("\t","",texto_columnas)
 texto_columnas<-gsub("\n","",texto_columnas)
 e<-which(texto_columnas=="")
 if (length(e)>0) {texto_columnas<-texto_columnas[-e]}
 e<-which(texto_columnas=="inc")
 if (length(e)>0) {texto_columnas<-texto_columnas[-e]}


 for (i in elimina) {
```

```
  e<-which(texto_columnas==i)
  if (length(e)>=1){
    texto_columnas<-texto_columnas[-e]
  }
 }

 caracteres<-nchar(texto_columnas)
 texto_columnas<-texto_columnas[which(caracteres>2)]

 texto_columnas<-wordStem(texto_columnas,language = "en")
 texto_columnas<-unique(texto_columnas)
 texto_columnas<-wordStem(texto_columnas,language = "es")
 texto_columnas<-unique(texto_columnas)

 b<-c()
 for (z in 1:length(texto_columnas)) {
  b<-paste(b,texto_columnas[z],sep = " ")
 }
 texto_columnas<-b
 texto_columnas<-trimws(texto_columnas)

}



#Creacion de campo texto_nuevo a partir del campo texto normalizado (sin palabras
irrelevantes)
texto_nuevo<-sapply(texto_split,limpio)
tipificaciones_resumen$texto_nuevo<-texto_nuevo


#Generación de diccionario de palabras (contiene todas las palabras distintas del campo
texto_nuevo)
lista_nuevo<-strsplit(texto_nuevo, split = " ")
```

```
texto_columnas<-unlist(lista_nuevo)
texto_columnas<-unique(texto_columnas)

#Nos quedamos con las plabras que tengan mas de 20 repeticiones
v<-c()
for (i in texto_columnas) {
 a<-0
 for (j in 1:n) {
  if(grepl(i,tipificaciones_resumen[j,'texto_nuevo'])){
   a<-a+1
  }
 }
 v[i]<-a
}

pesos<-data.frame(texto_columnas,v)
pesos$texto_columnas<-as.character(pesos$texto_columnas)
texto_columnas<-pesos$texto_columnas[(which(pesos$v>20))]

#Creamos la matriz a la que le vamos a aplicar Boosting. Tendra tantas variables como palabras tenga el diccionario.
#Si el registro contiene la palabra en el campo texto_nuevo aparecera un 1, si no un 0.
matriz<-data.frame()
for (i in texto_columnas) {
 for (j in 1:n) {
  a<-0
  if(grepl(i,tipificaciones_resumen[j,'texto_nuevo'])){
   a<-a+1
   matriz[j,i]<-a
  }else{
   matriz[j,i]<-a
```

```
    }
  }
}

#Añadimos a la matriz el campo tipificacion como factor
matriz$tipificacion<-factor(tipificaciones_resumen$Tipificacion)

save(matriz,file="MatrizTFG.RDat")
```

MODELOS

```
#TRABAJO FIN DE GRADO
#TECNICAS BOOSTING
#MANUEL PEREZ GARCIA
#DOBLE GRADO EN MATEMATICAS Y ESTADISTICA
#DEPARTAMENTO DE ESTADISTICA E INVESTIGACION OPERATIVA

##############################
#CONSTRUCCION DE LOS MODELOS
##############################



load(file="MatrizTFG.RDat")
dim(matriz)
table(matriz$tipificacion)
#Dividimos los datos en conjunto train y conjunto test
set.seed(25104)
n=nrow(matriz)
train = sample(n,n*3/4,replace = F)
datos_train<-matriz[train,]
table(datos_train$tipificacion)
filas_train<-nrow(datos_train)
datos_test<-matriz[-train,]
row.names(datos_train)<-1:filas_train

dim(datos_train)
dim(datos_test)

names(datos_train) <- make.names(names(datos_train))
names(datos_test)= names(datos_train)
```

```
#ADABOOST.M1: LIBRERIA ADABAG
########################################
library(adabag)
set.seed(45678)
error<-c()
for (i in 2:10) {
  print(i)
  modelo = boosting(tipificacion~.,data=datos_train,mfinal=50,
                    control = rpart.control(maxdepth =i))
  pred = predict.boosting(modelo,newdata=datos_test)
  error[i]<-100*pred$error
}
error<-error[-1]
error_min<-which.min(error)
plot(2:10,error,
     main = "Error por longitud de arbol",
     type = "l",
     xlab = "Longitud arboles",
     ylab = "Error clasificacion",
     col="blue")
points(error_min+1,error[error_min],
       type = "p",
       lwd=2,
       col="red")


#Adaboost definitivo
set.seed(45678)
modelo_M1 = boosting(tipificacion~.,data=datos_train,mfinal=50,
                     control = rpart.control(maxdepth =error_min+1))
```

```
save(modelo_M1,file = "Modelo_M1.Rdat")
pred = predict.boosting(modelo_M1,newdata=datos_test)
error_adaboostM1<-pred$error

#BAGGING
################################
set.seed(45678)
error<-c()
for (i in 2:10) {
  print(i)
  modelo = bagging(tipificacion~.,data=datos_train,mfinal=50,
                   control = rpart.control(maxdepth =i))
  pred = predict.bagging(modelo,newdata=datos_test)
  error[i]<-100*pred$error
}
error<-error[-1]
error_min<-which.min(error)
plot(2:10,error,
     main = "Error por longitud de arbol",
     type = "l",
     xlab = "Longitud arboles",
     ylab = "Error clasificacion",
     col="blue")
points(error_min+1,error[error_min],
       type = "p",
       lwd=2,
       col="red")


#Bagging definitivo
set.seed(45678)
```

```
modelo_bag = bagging(tipificacion~.,data=datos_train,mfinal=50,
                control = rpart.control(maxdepth =error_min+1))
save(modelo_bag,file = "Modelo_bag.Rdat")
pred = predict.bagging(modelo_bag,newdata=datos_test)
error_bagging<-pred$error


#GRADIENT BOOSTING: Libreria GBM
###################################
library(gbm)
cv_error  <- vector("numeric")
n_arboles <- vector("numeric")
shrinkage <- vector("numeric")

#Encontramos el mejor parametro de shrinkage
set.seed(12345)
for (i in c(0.001, 0.01, 0.1)) {
arbol_boosting <- gbm (tipificacion~.,
                  data = datos_train,
                  n.trees = 50,
                  n.minobsinnode = 5,
                  bag.fraction = 0.5,
                  cv.folds = 4)

cv_error  <- c(cv_error, arbol_boosting$cv.error)
n_arboles <- c(n_arboles, seq_along(arbol_boosting$cv.error))
shrinkage <- c(shrinkage, rep(i, length(arbol_boosting$cv.error)))
}
error <- data.frame(cv_error, n_arboles, shrinkage)

#Para obtener el mejor valor de shrinkage podemos estudiar la
```

```
#última iteración para cada valor candidato
error[c(50,100,150),]
shrinkage_valor=error[c(50,100,150),][which.min(error[c(50,100,150),1]),3]
shrinkage_valor
which.min(error[c(50,100,150),1])
ggplot(data = error, aes(x = n_arboles, y = cv_error,
                         color = as.factor(shrinkage))) +
  geom_smooth() +
  labs(title = "Evolución del cv-error", color = "shrinkage") +
  theme_bw() +
  theme(legend.position = "bottom")

dim(error)
#La linea roja es menor -> shrinkage=0.001


#Con el valor de shrinkage hallado, buscamos el mejor valor para el numero minimo de observaciones en un nodo
cv_error  <- vector("numeric")
n_arboles <- vector("numeric")
n.minobsinnode <- vector("numeric")
set.seed(23456)
for (i in c(1, 5, 10, 20)) {
  arbol_boosting <- gbm(tipificacion~.,
                        data = datos_train,
                        n.trees = 50,
                        shrinkage = shrinkage_valor,
                        n.minobsinnode = i,
                        bag.fraction = 0.5,
                        cv.folds = 4)
  cv_error  <- c(cv_error, arbol_boosting$cv.error)
  n_arboles <- c(n_arboles, seq_along(arbol_boosting$cv.error))
```

```
  n.minobsinnode <- c(n.minobsinnode,
                      rep(i, length(arbol_boosting$cv.error)))
}
error <- data.frame(cv_error, n_arboles, n.minobsinnode)
dim(error)
error[c(50,100,150,200),]
minobs=error[c(50,100,150,200),][which.min(error[c(50,100,150,200),1]),3]
minobs
ggplot(data = error, aes(x = n_arboles, y = cv_error,
                         color = as.factor(n.minobsinnode))) +
  geom_smooth() +
  labs(title = "Evolución del cv-error", color = "n.minobsinnode") +
  theme_bw() +
  theme(legend.position = "bottom")

#La linea roja parece menor -> n.minobsinnode=1

#Aumento numero arboles
set.seed(23456)
arbol_boosting <- gbm(tipificacion~.,
                      data = datos_train,
                      n.trees = 500,
                      shrinkage = shrinkage_valor,
                      n.minobsinnode = minobs,
                      bag.fraction = 0.5,
                      cv.folds = 4)
cv_error  <- arbol_boosting$cv.error
n_arboles <- 1:500
n.minobsinnode<-rep(minobs,500)
error <- data.frame(cv_error, n_arboles, n.minobsinnode)
ggplot(data = error, aes(x = n_arboles, y = cv_error,
```

```
                    color = as.factor(n.minobsinnode))) +
  geom_smooth() +
  labs(title = "Evolución del cv-error", color = "n.minobsinnode") +
  theme_bw() +
  theme(legend.position = "bottom")

#Procedimiento definitivo con los parametros hallados
arbol_boosting <- gbm(tipificacion~.,
                      data = datos_train,
                      n.trees = 500,
                      shrinkage = shrinkage_valor,
                      n.minobsinnode = minobs,
                      bag.fraction = 0.5,
                      cv.folds = 4)


save(arbol_boosting,file="ModeloGradientBoosting.RDat")
# Chequeamos el modelo sobre el testing data set
yhat.boost <- predict(arbol_boosting, newdata = datos_test, n.trees = 500)
y<-yhat.boost[,,1]
estimado<-c()
sol<-c()
for (i in 1:nrow(y)) {
  estimado[i]<-which(y[i,]==max(y[i,]))
  sol[i]<-colnames(y)[estimado[i]]
}

#Porcentaje de fallo
error_gbm<-1-mean(sol==datos_test$tipificacion)
table(sol)
```

```
#RANDOM FOREST
####################
set.seed(34567)
p<-ncol(datos_train)
acierto<-c()
library (randomForest)

#Eleccion del mejor modelo

for (i in 1:15) {
 print(i)
RF<-randomForest(tipificacion~.,
                 data = datos_train,
                 ntree = 500,
                 mtry = floor(sqrt(p)),
                 maxnodes=2^i
 )
 pred.rf<-predict(RF,datos_test,type = "prob")


estimado<-c()
sol<-c()
for (j in 1:nrow(pred.rf)) {
 estimado[j]<-which(pred.rf[j,]==max(pred.rf[j,]))
 sol[j]<-colnames(pred.rf)[estimado[j]]
}

acierto[i]<-mean(sol==datos_test$tipificacion)
}
```

```
acierto_max<- which.max(acierto)
plot(1:15,acierto,
    main = "Acierto por longitud de arbol",
    type = "l",
    xlab = "Longitud arboles",
    ylab = "Acierto clasificacion",
    col="blue")
points(acierto_max,acierto[acierto_max],
     type = "p",
     lwd=2,
     col="red")

#RF Definitivo
set.seed(34567)
RF<-randomForest(tipificacion~.,
              data = datos_train,
              ntree = 500,
              mtry = floor(sqrt(p)),
              maxnodes=2^acierto_max
)
pred.rf<-predict(RF,datos_test,type = "prob")
RF
plot(RF)

#Guardamos el modelo
save(RF,file="ModeloRF.RDat")

#Porcentaje de acierto
for (j in 1:nrow(pred.rf)) {
  estimado[j]<-which(pred.rf[j,]==max(pred.rf[j,]))
  sol[j]<-colnames(pred.rf)[estimado[j]]
```

```
}
mean(sol==datos_test$tipificacion)
table(sol)
error_rf<-1-mean(sol==datos_test$tipificacion)



#COMPARACION DE MODELOS
#########################
cbind(error_adaboostM1,error_bagging,error_gbm,error_rf)
```

- Scripts Datos Carseats

```
#TRABAJO FIN DE GRADO
#TECNICAS BOOSTING
#MANUEL PEREZ GARCIA
#DOBLE GRADO EN MATEMATICAS Y ESTADISTICA
#DEPARTAMENTO DE ESTADISTICA E INVESTIGACION OPERATIVA

##############################
#CONSTRUCCION DE LOS MODELOS
##############################

library(adabag)
library(gbm)
library(ISLR)
library(randomForest)

attach(Carseats )
High=ifelse(Sales <=8,"No","Yes ")

matriz<-data.frame(Carseats,High)
summary(matriz)
table(matriz$High)

#Dividimos los datos en conjunto train y conjunto test
set.seed(25104)
n<-nrow(matriz)
train <- sample(n,n*3/4,replace = F)
datos_train<-matriz[train,]
filas_train<-nrow(datos_train)
datos_test<-matriz[-train,]
```

```
dim(datos_train)
dim(datos_test)

#ADABOOST.M1: LIBRERIA ADABAG
########################################

#Boosting
porc_error<-c()
tamaño_arbol<-1:10
set.seed(12345)
for (i in tamaño_arbol) {
  modelo = boosting(High~.-Sales,data=datos_train,mfinal=50,
                    control = rpart.control(maxdepth =i))
  pred = predict.boosting(modelo,newdata=datos_test)
  porc_error[i]<-100*pred$error

}
tamaño<-which(porc_error==min(porc_error))
plot(tamaño_arbol,porc_error,
     main="Evolución error",
     xlab="Tamaño Arbol",
     ylab="Porcentaje error",
     type="l")
points(tamaño,porc_error[tamaño],
       type = "p",
       lwd=2,
       col="red")


#Definitivo Boosting
```

```
set.seed(12345)
modelo = boosting(High~.-Sales,data=datos_train,mfinal=50,
                  control = rpart.control(maxdepth =tamaño[1]))

#Predicciones
pred.boost<-predict.boosting(modelo,newdata=datos_test)
pred.boost$confusion
error_boost<-pred.boost$error

#Curva ROC
library(ROCR)

pred.test1<-pred.boost$prob[,2]
pred1<-prediction(pred.test1,datos_test$High)

perf_auc1<-performance(pred1,"auc")
auc_test<-perf_auc1@y.values[[1]]
auc_test

perf_ROC1<-performance(pred1,"tpr","fpr")
plot(perf_ROC1,main="ROC Plot test")
abline(0,1)

save(modelo,file="ModeloAdaBoostM1_CARSEATS.RDat")

#BAGGING
##########################################
porc_error<-c()
tamaño_arbol<-1:10
set.seed(12345)
for (i in tamaño_arbol) {
```

```
  modelo = bagging(High~.-Sales,data=datos_train,mfinal=50,
                   control = rpart.control(maxdepth =i))
  pred = predict.bagging(modelo,newdata=datos_test)
  porc_error[i]<-100*pred$error

}
tamaño<-which(porc_error==min(porc_error))

plot(tamaño_arbol,porc_error,
     main="Evolución error",
     xlab="Tamaño Arbol",
     ylab="Porcentaje error",
     type="l")
points(tamaño,porc_error[tamaño],
       type = "p",
       lwd=2,
       col="red")


#Definitivo Bagging
set.seed(12345)
modelo = bagging(High~.-Sales,data=datos_train,mfinal=50,
                 control = rpart.control(maxdepth =tamaño[1]))

#Predicciones
pred.bag<-predict.bagging(modelo,newdata=datos_test)
pred.bag$confusion
error_bag<-pred.bag$error

#Curva ROC
library(ROCR)
```

```
pred.test1<-pred.bag$prob[,2]
pred1<-prediction(pred.test1,datos_test$High)

perf_auc1<-performance(pred1,"auc")
auc_test<-perf_auc1@y.values[[1]]
auc_test

perf_ROC1<-performance(pred1,"tpr","fpr")
plot(perf_ROC1,main="ROC Plot test")
abline(0,1)

save(modelo,file="ModeloBagging_CARSEATS.RDat")



#RANDOM FOREST
############################################
names(datos_train) <- make.names(names(datos_train))
names(datos_test)= names(datos_train)
p<-ncol(datos_train)
error<-c()
library (randomForest)
library(ROCR)
set.seed(12345)
for (i in 1:5) {
  RF<-randomForest(High~.-Sales,
                   data = datos_train,
                   ntree = 500,
                   mtry = floor(sqrt(p)),
                   maxnodes=2^i
```

```
  )
  pred.rf<-predict(RF,datos_test,type = "prob")[,2]
  pred1<-prediction(pred.rf,datos_test$High)
  perf_auc1<-performance(pred1,"auc")
  auc_test<-perf_auc1@y.values[[1]]
  error[i]<-auc_test
}

maxnode<-which.max(error)
plot(1:5,error,type="l",ylab="AUC",main="AUC por numero de nodos")
points(maxnode,error[maxnode],
      type = "p",
      lwd=2,
      col="red")

#Definitivo
set.seed(12345)
RF<-randomForest(High~.-Sales,
                 data = datos_train,
                 ntree = 500,
                 mtry = floor(sqrt(p)),
                 maxnodes=2^maxnode[1])

RF
plot(RF)

pred.rf<-predict(RF,data=datos_test)



#Ahora calcular rendimiento
```

```
pred.rf<-predict(RF,datos_test,type = "prob")[,2]
pred1<-prediction(pred.rf,datos_test$High)

perf_auc1<-performance(pred1,"auc")
auc_test<-perf_auc1@y.values[[1]]
auc_test

perf_ROC1<-performance(pred1,"tpr","fpr")
plot(perf_ROC1,main="ROC Plot test")
abline(0,1)

save(RF,file="ModeloRF_CARSEATS.RDat")

pred.rf<-predict(RF,datos_test,type = "class")
error_rf<-1-sum(pred.rf==datos_test$High)/nrow(datos_test)
table(pred.rf,datos_test$High)


#COMPARACION DE MODELOS
#########################
cbind(error_boost,error_bag,error_rf)
```

9. BIBLIOGRAFÍA

- Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A. (1984). *Classification and Regression Trees*.
- Trevor Hastie, Robert Tibshirani, Jerome Friedman. *The Elements of Statistical Learning Data Mining, Inference, and Prediction*. 2ª Ed. New York: Springer; 2001.
- Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R*. New York : Springer, 2013.
- Natekin, Alexey & Knoll, Alois. (2013). *Gradient Boosting Machines, A Tutorial*. Frontiers in neurorobotics. 7. 21. 10.3389/fnbot.2013.00021.
- Greg Ridgeway with contributions from others (2017). gbm: Generalized Boosted Regression Models. R package version 2.1.3. https://CRAN.R-project.org/package=gbm
- Kenneth Benoit, David Muhr and Kohei Watanabe (2017). stopwords: Multilingual Stopword Lists. R package version 0.9.0. https://CRAN.R-project.org/package=stopwords
- Milan Bouchet-Valat (2014). SnowballC: Snowball stemmers based on the C libstemmer UTF-8 library. R package version 0.5.1. https://CRAN.R-project.org/package=SnowballC
- R Special Interest Group on Databases (R-SIG-DB), Hadley Wickham and Kirill Müller (2018). DBI: R Database Interface. R package version 1.0.0. https://CRAN.R-project.org/package=DBI
- Jeroen Ooms, David James, Saikat DebRoy, Hadley Wickham and Jeffrey Horner (2018). RMySQL: Database Interface and 'MySQL' Driver for R. R package version 0.10.15. https://CRAN.R-project.org/package=RMySQL
- Gagolewski M. and others (2018). R package stringi: Character string processing facilities. http://www.gagolewski.com/software/stringi/. DOI:10.5281/zenodo.32557
- Alfaro, E., Gamez, M. Garcia, N.(2013). adabag: An R Package for Classification with Boosting and Bagging. Journal of Statistical Software, 54(2), 1-35. URL http://www.jstatsoft.org/v54/i02/.
- A. Liaw and M. Wiener (2002). Classification and Regression by randomForest. R News 2(3), 18--22.
- Gareth James, Daniela Witten, Trevor Hastie and Rob Tibshirani (2017). ISLR: Data for an Introduction to Statistical Learning with Applications in R. R package version 1.2. https://CRAN.R-project.org/package=ISLR

- Sing T, Sander O, Beerenwinkel N, Lengauer T (2005). “ROCR: visualizing classifier performance in R.” _Bioinformatics_, *21*(20), 7881. <URL: http://rocr.bioinf.mpi-sb.mpg.de>.