Universidade Vigo

Trabajo Fin de Máster

Métodos para la mejora de predicciones en clases desbalanceadas en el estudio de bajas de clientes (CHURN)

Hugo Antonio Arnejo Calviño

Máster en Técnicas Estadísticas

Curso 2016-2017

Propuesta de Trabajo Fin de Máster

Título en galego: Métodos para a mellora de prediccins en clases desbalanceadas no estudo de baixas de clientes (CHURN)

Título en español: Métodos para la mejora de predicciones en clases desbalanceadas en el estudio de bajas de clientes (CHURN)

English title: Predictive models for imbalanced data

Modalidad: Modalidad B

Autor/a: Hugo Antonio Arnejo Calviño, Universidad de Santiago de Compostela

Director/a: Javier Roca Pardiñas, Universidad de Vigo

Tutor/a: Jose Ramón Sousa Vázquez, Optare Solutions

Breve resumen del trabajo:

Analizar métodos que permitan mejorar las predicciones en clases desbalanceadas en estudios de bajas de clientes (CHURN) en operadoras de telefonía. Para ello seguiremos el siguiente guión que se corresponde con la memoria del trabajo:

- Introducción
- Descripción del problema
- Técnicas empleadas
- Aplicación a datos reales
- Conclusiones

Recomendaciones:

Haber cursado las materias del Máster en Técnicas Estadísticas: Series de Tiempo y Estadística no Paramétrica. Conocimientos del lenguaje estadístico de programación R.

Don Javier Roca Pardiñas, profesor de la Universidad de Vigo, y don Jose Ramón Sousa Vázquez, director de tecnologías OSS de Optare Solutions, informan que el Trabajo Fin de Máster titulado

Métodos para la mejora de predicciones en clases desbalanceadas en el estudio de bajas de clientes (CHURN)

fue realizado bajo su dirección por don Hugo Antonio Arnejo Calviño para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, dan su conformidad para su presentación y defensa ante un tribunal.

En Santiago, a 8 de Noviembre de 2016.

El director:

Don Javier Roca Pardiñas

El tutor:

Don Jose Ramón Sousa Vázquez

El autor:

Don Hugo Antonio Arnejo Calviño

Agradecimientos

En primer lugar me gustaría dar las gracias a Optare Solutions por brindarme la oportunidad de hacer las prácticas de fin de máster con ellos, así como a todos sus miembros por la buena acogida y el trato recibido durante mi estancia.

En segundo lugar, agradecer a Javier Roca Pardiñas y Jose Ramón Sousa Vázquez todas las indicaciones recibidas para la confección de este trabajo.

Además, me gustaría hacer una mención especial a David Lozano Núñez e Irene Castro Conde por su ayuda desinteresada desde el primer momento y de los que me llevo una gran amistad.

Por último, muchas gracias a mi familia y amigos por todo el apoyo recibido, especialmente en los últimos meses.

Índice general

Resumen

Resumen en español

La presencia de clases desbalanceadas en un conjunto de datos supone un problema en los modelos predictivos ya que éstos tienden a centrar su atención sobre los casos de la clase mayoritaria a la hora de realizar las predicciones. En este trabajo estudiaremos varias de las técnicas que se han ido desarrollando recientemente para hacer frente a esta cuestión y de como aplicarlas en un caso concreto como la gestión del churn. Comenzaremos por una breve introducción que nos permita contextualizar el problema del desbalanceo y ya en el segundo capítulo, abordaremos este concepto de forma más detallada, además de introducir las primeras técnicas que nos permitan subsanarlo de cara a los modelos de predicción. En el tercer capítulo desarrollaremos en profundidad cada una de las técnicas que en este trabajo se presentan, razonando los motivos que nos han llevado a elegirlas, tanto a nivel teórico como práctico. A continuación, en la cuarta sección mostraremos los resultados obtenidos para cada una de estas técnicas así como los criterios de evaluación que nos permitirán establecer una comparativa entre ellas. Finalmente, expondremos una serie de conclusiones que servirán para cohesionar todas las ideas plasmadas a lo largo de este trabajo.

English abstract

The presence of unbalanced classes in a data set can be a problem in predictive models as they tend to focus on the cases of the majority class. In this paper, we will study a few techniques that have been recently implemented to address this issue and how to apply this into questions as churn it is. The first chapter begins with a brief introduction that allows us to contextualize the proposed problem. In the second section, we will make a description of the problem introducing the concept of unbalanced data and the first techniques to correct it ahead prediction models. In the third chapter we will develop each of the techniques presented in this paper and we'll also state the reasons that led us to choose them, both theoretical and practical level. Chapter four shows the results obtained by each of these techniques and evaluation criteria to make a comparison between them. Finally, we will show a couple of conclusions in order to unify all the ideas that we checked throughout this paper.

Prefacio

La minería de datos (en inglés, *data mining*) es una rama de la estadística y las ciencias de la computación referida al proceso que intenta descubrir patrones en grandes volúmenes de conjuntos de datos y, en este campo, el desbalanceo de clases ha sido una de los grandes retos a resolver. Este concepto hace alusión al desequilibrio que existe en una muestra de datos cuando el número de instancias de un grupo (clase mayoritaria) es superior al de otros grupos (clases minoritarias). Muchos de los métodos de clasificación convencionales tienden a centrarse sobre la clase mayoritaria ignorando la minoritaria. De hecho, sus clasificaciones suelen verse afectadas de forma negativa por este desajuste en la distribución de los datos. Con el objetivo de corregir el problema del desbalanceo, los investigadores han ido desarrollando varias técnicas y algoritmos que podemos clasificar en dos grupos: técnicas predictivas (aprendizaje supervisado) y técnicas descriptivas (aprendizaje no supervisado). En nuestro caso, abordaremos las técnicas correspondientes al primer grupo ya que obtendremos modelos capaces de predecir el valor correspondiente a un objeto después de haber analizado los datos de entrenamiento y en donde la variable respuesta es conocida. En particular, llevaremos a cabo una tarea de clasificación ya que se pretende predecir el estado final de cada cliente (baja/no baja).

Todas las técnicas que se han ido proponiendo comparten el mismo objetivo pero su funcionamiento es distinto, sobre todo a nivel de los datos. Algunas de las que en este trabajo se presentan realizan un preprocesado de la muestra (es decir, modifican su distribución) antes de aplicar el modelo clasificatorio, como por ejemplo los algoritmos de undersampling [31] u oversampling [11]. Por otro lado, se abordarán otras técnicas que no realizan este análisis previo y que, por tanto, no atacan de forma directa el problema del desbalanceo aunque son capaces de mejorar los resultados, como pueden ser los métodos bagging [47], boosting [2] o Random Forest [30].

La combinación de clasificadores ha sido un método muy empleado recientemente para la mejora en la clasificación. En 1994, el estadístico Leo Breiman hace uso de esta idea para proponer el método bagging [5], donde nos muestra que si las perturbaciones en la muestra de datos causan cambios significativos en el predictor construido, entonces el método bagging puede mejorar la precisión del mismo. Una de las modificaciones de este método es el algoritmo Random Forest, cuya versión más moderna se desarrolla en el año 2001 por el propio Breiman [6] y la profesora del Departamento de Matemáticas y Estadística de Utah, Adele Cutler. Por otro lado, el boosting se basa en el problema propuesto por los científicos de la computación, Michael Kearns y Leslie Valiant, que plantean la siguiente cuestión: ¿puede un conjunto de clasificadores crear un clasificador más fuerte? [27][28], y que encuentra respuesta un año más tarde de la mano del profesor Robert E. Schapire [46], el cual es capaz de construir un modelo más preciso a partir de los estudios previos realizados por Kearns y Valiant.

Este análisis sobre las técnicas que permiten corregir el problema del desbalanceo para las predicciones del churn tiene el fin de transmitir la máxima información al cliente por lo que nuestro objetivo no será la optimización de estos algoritmos, sino estudiar el comportamiento de los clasificadores ante el problema del desbalanceo de clases para poder seleccionar aquellos que proporcionen los mejores resultados y que éstos sean fácilmente interpretables. Por eso, la mayor parte de ellos se construirán a partir de los árboles de decisión ya que nos permiten obtener las reglas por las cuales se asignan las probabilidades de baja a cada uno de los clientes.

En el primer capítulo haremos una breve introducción en donde describiremos la empresa que ha dado pié a este proyecto, indicando el origen y los objetivos del mismo. A continuación, pasaremos a la descripción del problema sobre el que gira este trabajo, el desbalanceo de clases, así como las distintas técnicas que se estudiarán para hacerle frente y que en la tercera sección se muestran con detalle. Además, ofreceremos una aplicación práctica de todo lo expuesto en el cuarto capítulo, en donde describimos el escenario ante el que nos encontraremos, cómo procederemos en la evaluación de los modelos predictivos que se irán construyendo y los resultados que nos van a proporcionar cada uno de ellos. Finalmente, cerraremos con una serie de conclusiones que nos permitirán unificar y comprender las ideas expuestas en el presente trabajo.

Para un correcto seguimiento del trabajo es recomendable tener conocimientos en las siguientes materias del Máster en Técnicas Estadísticas: Series de Tiempo y Estadística no Paramétrica.

Capítulo 1

Introducción

En este primer capítulo contextualizaremos el problema del desbalanceo de clases. Para ello, comenzaremos por hacer una leve descripción de la empresa que ha colaborado en este proyecto, Optare Solutions, así como el proyecto en el que se encuentra inmerso y que engloba todo el estudio que emprenderemos de aquí en adelante. A continuación, mostraremos todo aquello que nos va a permitir la revisión de las técnicas que se irán viendo a lo largo del presente proyecto apoyándonos en argumentos sólidos que justifiquen la selección de las mismas.

1.1. Optare Solutions

Optare Solutions S.L.[1] es una consultora tecnológica especializada en la implantación, mantenimiento, desarrollo e integración de sistemas BSS (Business Support Systems), en la realización de OSS (Operations Support Systems) y en la prestación de servicios profesionales de consultoría en el ámbito de los operadores de Telecomunicaciones, que está situada en el Parque Tecnológico y Logístico de Valladares, Vigo.

Nació en el año 2002 con el objetivo de proporcionar consultoría técnica para la provisión de servicios complejos en una operadora emergente del mercado español, ayudando a que sus sistemas soportasen un gran crecimiento de su volumen de negocio (más de 10 veces en 3 años). Durante estos años otras operadoras han requerido de sus servicios y han continuado ayudando a sus clientes a crecer, mejorar su eficiencia, reducir costes y convertirse en empresas más ágiles. Desde el año 2011, Optare Solutions cuenta además con una división de consultoría internacional proporcionando formación y servicios de los productos de Oracle Communications específicos para el sector telecom.

La experiencia, el conocimiento y los resultados obtenidos en los más de 13 años de existencia de Optare Solutions les han hecho merecedores del reconocimiento internacional como "*Cool Vendor in Telecom Operations Management System* 2010"por Gartner Inc, firma americana líder en análisis del mercado TIC (Tecnologías de la Información y Comunicación).

1.2. Origen del proyecto y objetivos

Este proyecto representa una continuación del trabajo sobre los resultados obtenidos tras el proyecto BDA4T (*Big Data Analytics for Telecoms*) encaminado a la investigación y desarrollo, y que está financiado por el Centro para el Desarrollo Tecnológico Industrial (CDTI), contando con el apoyo de la Universidad de Vigo como organismo de investigación.

---

[1] *http* : *//optaresolutions.com/es/*

Este proyecto se basó en el desarrollo de cuatro líneas principales de investigación:

- Sistemas de almacenamiento NoSQL (Not only Structured Query Language) donde se estudió la adaptación de los sistemas NoSQL para su uso con los datos internos de los operadores de forma que sea posible almacenarlos de forma eficiente.
- Plataforma de analítica avanzada que permita la realización de "Prescriptive Analytics" para los operadores de telecomunicaciones.
- Automatización de acciones telecom en donde se desarrolló un módulo que permitía definir acciones automatizadas en función de los resultados analíticos obtenidos.
- Análisis del valor añadido del sistema BDA4T: se analizó la mejora que obtienen los operadores con el uso de los resultados de este proyecto.

Optare Solutions participó activamente en todas las líneas de investigación de este proyecto y lideró el desarrollo de los siguientes casos de uso:

- **Cálculo del Customer Lifetime Value:** en este caso de uso se estudiaron métodos para estimar el tiempo de vida y los ingresos futuros de los clientes que permitan conocer su valor para el operador. Con estos datos, el operador puede priorizar la gestión de estos clientes para mejorar su satisfacción e incrementar sus resultados.
- **Modelo predictivo de bajas:** con cifras récord de portabilidades en España mes a mes, las bajas son uno de los mayores problemas que tienen los operadores para mejorar sus resultados. En este caso de uso se estudiaron y desarrollaron diferentes modelos de predicción para determinar la probabilidad de baja de un cliente según su perfil comercial que permitan a los operadores centrar sus esfuerzos de fidelización en los clientes más rentables y con mayor probabilidad de abandonar la compañía.
- **Modelado de QoE (Quality of Experience):** estudio de la conversión de las medidas de QoS (Quality of Service) de las que dispone internamente el operador en sus sistemas en medidas de QoE que permitan evaluar la satisfacción de los clientes con el servicio proporcionado por los operadores.

El modelado predictivo de bajas será el principal punto sobre el que se asentará todo el estudio que realizaremos en este trabajo. Construiremos varios modelos para predecir las bajas de los operadores empleando los datos proporcionados por sus propios sistemas. El problema reside en la naturaleza de los datos con los que vamos a trabajar, el desbalanceo de las clases, cuestión que en el siguiente capítulo explicaremos con detalle. Nuestro objetivo será estudiar las técnicas que permitan hacer estas predicciones de la forma más fiable posible atendiendo al problema del desbalanceo y obteniendo un número razonable de bajas predichas. Por ejemplo, si contáramos con una cartera de 100 clientes en donde tenemos una tasa del baja del 2 %, buscaríamos construir modelos que predigan un porcentaje de baja en torno a éste, es decir, en torno a 2 personas.

1.3. ¿Qué es el churn?

El hecho de que un cliente abandone la compañía presenta un efecto temporal y los factores que contribuyen a él están generalmente en un estado transitorio ya que la idea que lleva a dicho cliente a plantearse su marcha puede ser diferente mañana y seguro que será distinta dentro de un cierto período de tiempo. Este hecho suele pasarse por alto en los operadores y es lo que hace del modelo probabilista una necesidad.

La construcción de estos modelos de predicción resultan muy útiles ya que nos van a permitir varias cosas:

- Estimar el tiempo que el cliente permanece en la compañía así como sus ingresos futuros.
- Predecir la probabilidad de baja de un cliente según su perfil para que los operadores puedan desarrollar campañas de retención sobre aquellos clientes con mayor probabilidad de baja.

En este trabajo nos centraremos más en el segundo apartado ya que los modelos que vamos a construir proporcionarán una predicción del *churn*. Este término hace alusión a la tasa de clientes que abandonan la compañía o al proveedor de un servicio durante un período de tiempo determinado y describe la infidelidad o falta de lealtad de los clientes. Se cuantifica en tanto por ciento y se calcula como sigue:

$$\text{Churn} = \frac{(\text{Nº de clientes al principio} - \text{Nº de clientes al final})}{\text{Nº de clientes al comienzo del periodo}} \cdot 100$$

Los períodos más utilizados para el cálculo de la tasa de abandonos son mensual, trimestral y anual. Por otro lado, el número de clientes perdidos en un período determinado se obtiene sumando la cartera activa al comienzo de dicho período y los clientes dados de alta en ese período; y posteriormente restándole la cartera activa al finalizar dicho período, como se muestra a continuación:

$$\text{Clientes perdidos} = (\text{Cartera activa inicial} + \text{Clientes dados de alta}) - \text{Cartera activa al final.}$$

Hemos de destacar que resulta entre cinco y quince veces más caro captar nuevos clientes que retener a los actuales, por eso tiene más sentido centrarse en promover una campaña de retención de clientes que en diseñar una para captar otros nuevos.

Generalmente se distinguen dos tipos de churn: voluntario e involuntario. El churn voluntario se produce cuando el cliente decide por sí mismo cambiarse de compañía, mientras que el churn involuntario se produce en casos como la reubicación del cliente en otra zona geográfica, la caída por morosidad o, en un caso extremo, por fallecimiento. En la mayoría de las aplicaciones de inteligencia de negocios el churn involuntario se excluye de los modelos predictivos, concentrándose principalmente en el churn voluntario, ya que éste, normalmente, es debido a factores de relación empresa-cliente.

Independientemente de todas las formas existentes para clasificar información, es importante considerar que las quejas, cambios de precio o la frecuencia de compra son factores a considerar en los movimientos del churn. Algunas empresas consideran que el precio es una de las principales causas de abandono despreciando que el servicio al cliente pueda llegar a ser una causa más importante que el precio.

1.3.1. El churn en España

A partir del informe Económico de las Telecomunicaciones y del Sector Audiovisual 2015 ilustrado en la Figura 1.1, podemos observar que el churn medio en España ha aumentado entre los distintos operadores de telefonía móvil. En particular, en el año 2014 se ha registrado una tasa de abandono del 29,1 %, lo que significa que cualquier operador móvil en España perdería todos sus clientes en poco más de tres años si no fuera capaz de captar nuevos clientes.

Por otro lado, se observa una cierta mejoría en los últimos años, fruto de la ligera recuperación económica actual así como los análisis realizados por los profesionales que informan debidamente a los operadores para poder emprender las acciones convenientes sobre sus clientes.

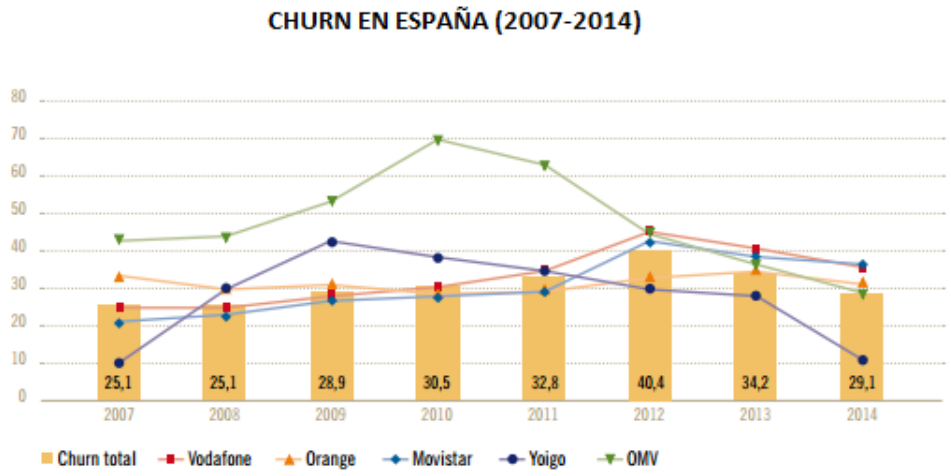


Figura 1.1: Churn total

Debido a la situación de crisis actual, los operadores han sufrido un ajuste en sus ingresos propiciado fundamentalmente por el descenso en el consumo de sus clientes, a pesar de que las comunicaciones son un bien esencial en la sociedad moderna. Los datos proporcionados por la Comisión Nacional de los Mercados y la Competencia (CNMC)[2] así lo revelan y puede verse reflejado en los ingresos de los operadores correspondientes a los últimos 6 años y que recogemos en la tabla 1.2. En ella podemos apreciar serios descensos en los ingresos a partir del año 2009 con una bajada en torno al 7 % y de forma continuada pero en diferente medida, hasta el año 2014.

2008	2009	2010	2011	2012	2013	2014
44136.82	41227.92	39799.92	37947.91	35629.93	32844	30889.92
0,6 %	−6,6 %	−3,5 %	−4,7 %	−7,1 %	−6,9 %	−6,0 %

Tabla 1.2. Ingresos totales anuales (millones de euros) de los operadores de telefonía móvil y porcentaje de cambio año a año.

Una buena forma de contribuir con este tipo de información, es realizar un estudio de su base de clientes a través de técnicas de analítica avanzada, con el fin de proporcionarles información sobre aquellos clientes que posean una mayor probabilidad de darse de baja para que puedan realizar acciones de fidelización de forma prioritaria sobre ellos. En este trabajo realizaremos un estudio que permite hacer este tipo de actuación aunque veremos que no todas las técnicas van a tener la misma incidencia en cuanto a la calidad de la información.

[2] *https : //www.cnmc.es/*

Capítulo 2

El desbalanceo de clases

En este capítulo empezaremos por hacer una introducción al problema de desbalanceo abordando dicho concepto y los distintos métodos de resolución que existen para poder solucionarlo. Además desarrollaremos el proceso de modelado que vamos a llevar a cabo para construir nuestros clasificadores que, en general, se basarán en la construcción de árboles de decisión que será el cierre de esta sección y que dará paso a las técnicas de desbalanceo que revisaremos en este trabajo.

2.1. Descripción del problema

El desbalanceo de clases es una característica de la muestra que se produce cuando una o más clases (clases minoritarias) se encuentran representadas en menor medida que otras (clases mayoritarias). Este fenómeno puede ocurrir a diferentes niveles y a día de hoy no existe un umbral que nos indique cuando una base de datos está desbalanceada. Esta cuestión supone un problema en el ámbito de la clasificación de nuevos datos ya que puede derivar en un deterioro notable en la eficiencia de nuestro clasificador. En particular, desempeñará una buena labor sobre la clase mayoritaria en detrimento de la minoritaria ya que detecta una mayor presencia en la muestra del primer grupo decantando la balanza sobre la clase mayoritaria a la hora de clasificar. Por ejemplo, si en una clase de 25 alumnos, 24 de ellos son rubios, probablemente si entra un alumno nuevo en la clase, nos atreveremos a decir que también es rubio. Desgraciadamente, esta suposición es un gran error por nuestra parte.

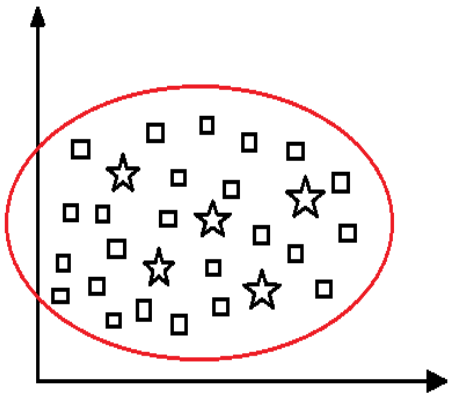


Figura 2.1: Gráfico de una distribución desbalanceada

De la Figura 2.1 se desprende, de una manera gráfica, el problema del desbalanceo de datos: nos encontramos ante un número reducido de ejemplos de estrellas (clase minoritaria) en comparación con el número de ejemplos correspondientes a los cuadrados (clase mayoritaria).

En la actualidad, el escenario de una distribución de datos desbalanceados se refleja en varias situaciones como por ejemplo la detección de fraudes [25], la prevención de intrusiones [12], las finanzas [29] o incluso, en investigaciones médicas [21].

Sin embargo, en muchos de los problemas en los que el desbalanceo de clases está presente, la clase minoritaria suele ser la de mayor interés, de modo que clasificar erróneamente casos pertenecientes a dicha clase supone un alto coste.

2.2. Métodos de resolución

El problema del desbalanceo de clases es una cuestión que se está abordando en la actualidad de forma activa y son muchos los investigadores que estudian y proponen nuevas técnicas para poder hacerle frente. La mayoría de ellos solo se han concentrado en resolver problemas que tienen que ver con dos clases, por ejemplo, se pueden definir casos con pacientes con enfermedades raras y personas que padecen otro tipo de enfermedades. Nuestro objetivo será construir modelos que sean capaces de predecir la baja de clientes y atendiendo a la información de la que se dispone, podemos distinguir dos tipos de técnicas:

- **Aprendizaje supervisado (técnicas predictivas):** está encaminado a la obtención de un modelo capaz de predecir el valor correspondiente de cualquier objeto (en nuestro caso, bajas de clientes) después de haber analizado la muestra de entrenamiento, dónde la variable respuesta es conocida. Si lo que se pretende es predecir una variable discreta basándonos en otros atributos del conjunto de datos (por ejemplo, el churn) se llama clasificación, y si predecimos una variable continua se llamará regresión (por ejemplo, el número de bajas).
- **Aprendizaje no supervisado (técnicas descriptivas):** son aquellos que realizan el ajuste de un modelo sin conocer datos a priori. Algunas de las más conocidas son:
  - **Análisis Clúster**: [15] dividimos los datos en grupos de elementos que tienen propiedades similares entre sí y diferentes a las de los otros grupos.
  - **Anaísis de Componentes Principales (ACP)**: [8] es una técnica utilizada para reducir la dimensionalidad de un conjunto de datos.
  - **Estudio de Correlaciones**: [39] busca detectar dependencias entre las variables.
  - **Detección de Anomalías**: [35] identifica casos inusuales para investigarlos en profundidad.
  - **Reglas de Asociación**: [9] buscan relaciones interesantes entre las variables de un conjunto de datos.
  - **Minería de texto**: [18] están dirigidos a la obtención de información interesante de un texto plano, como patrones que se repiten o análisis de sentimientos.

En este trabajo abordaremos las técnicas predictivas basadas en aprendizaje supervisado ya que disponemos de la información proporcionada por las operadoras de telefonía móvil, además de conocer los posibles valores de nuestra variable respuesta (baja/no baja). Los métodos de clasificación supervisada aparecen en casos reales como la detección de fraudes. Por ejemplo, un 99 % de las compras son correctas y un 1 % son fraudulentas. Si construimos un modelo, podemos estar seguros al 99 % de que todas nuestras compras serán correctas debido a la escasa presencia de un número de compras incorrectas. La cuestión es cómo podemos construir un modelo que sea capaz de detectar este número reducido de casos que constituyen la clase minoritaria.

Basándonos en la distinción realizada por la gran mayoría de la comunidad investigadora, existen dos metodologías o maneras de afrontar el desbalanceo de datos:

1. **Técnicas de remuestreo:** se basan en modificar la distribución inicial de los datos para balancear las clases. Algunas de las más importantes:

   - **Oversampling**: Consiste en modificar la distribución de los datos incrementando el número de casos de la clase minoritaria.
   - **Undersampling**: Consiste en modificar la distribución de los datos reduciendo el número de casos de la clase mayoritaria.
   - **Algoritmos híbridos**: Combinaremos las técnicas de undersampling y oversampling.

   Existen numerosos artículos que realizan un estudio sobre las distintas técnicas de remuestreo, como por ejemplo [17], [11], [13] o [24] entre otros.

2. **Modificación de algoritmos:** consiste en variar los algoritmos existentes para mejorar la predicción.

Los dos enfoques son independientes entre sí y pueden combinarse para mejorar el rendimiento de cada uno.

2.3. Modelado

Los argumentos para el correcto modelado de cada uno de los clasificadores se irán detallando en la sección 3 de este trabajo y aunque presentan características diferentes, a nivel funcional poseen un comportamiento similar y que podemos resumir en los siguientes pasos:

1. **Pre-procesado de los datos:** En algunos de los métodos que veremos se realiza una previa modificación de la distribución de los datos, por ejemplo, aumentando el número de casos de la clase minoritaria o reduciendo las instancias de la clase mayoritaria.

2. **Definición de la fórmula del modelo:** Al llevar a cabo la construcción de un modelo predictivo de bajas, fijaremos la variable "*Baja.objetivo*" como respuesta y las demás como variables explicativas. Se trata de una variable discreta que toma dos posibles valores:

   - SI = "El cliente se da de baja"
   - NO = "El cliente no se da de baja"

3. **Aplicación del modelo:** Cada una de los métodos que vamos a revisar tiene su propia metodología e implementación a nivel práctico. Ambas cuestiones serán abordadas de forma detallada en el siguiente capítulo.

Dentro de los modelos de predicción, los árboles de decisión [43] o *decision trees* son los más utilizados por una serie de razones que los hacen especialmente atractivos como: la sencillez del modelo, la amplitud de implementaciones que existen, la rapidez de clasificación de nuevos patrones, la posibilidad de representarlos gráficamente aportando así una explicación de la división efectuada, la fácil interpretación (en caso de no ser excesivamente grandes) y la posibilidad de obtener las reglas por las cuales asignamos cada una de las probabilidades de baja de los clientes de la compañía.

Todos estos motivos nos proporcionan un alto grado de comprensión del conocimiento utilizado en la toma de decisiones. En la siguiente sección profundizamos más sobre este concepto y expondremos algunos de los algoritmos más empleados en la actualidad [25].

2.3.1. Árboles de decisión

Para tomar una decisión, es necesario conocer, comprender y analizar el problema en cuestión, para así poder darle solución. En algunos casos, por ser tan simples y cotidianos, este proceso se realiza de forma implícita y se soluciona muy rápidamente, pero existen otros casos en los cuales, las consecuencias de una mala o buena elección pueden tener repercusiones en la vida y si es en un contexto laboral, en el éxito o fracaso de la organización. Esto justifica la necesidad de realizar un proceso más estructurado que pueda dar más seguridad e información para resolver el problema.

Descripción

Un árbol de decisión es una forma gráfica y analítica de representar todos los posibles sucesos que pueden ocurrir a partir de una decisión asumida en un momento concreto. Además, nos ayudan a tomar la decisión más "acertada"desde un punto de vista probabilístico, ante un abanico de posibles decisiones. Por otro lado permite desplegar visualmente un problema y organizar el trabajo de cálculo que se deba realizar.

Los árboles de decisión constituyen una técnica estadística para la segmentación, la estratificación, la predicción, la reducción de datos, el filtrado de variables, la identificación de interacciones, la fusión de categorías y la discretización de variables continuas. Estos múltiples usos han beneficiado a los analistas en la necesidad de describir condiciones y acciones así como las consecuencias que se deben considerar en la toma de decisiones. A pesar de esto, los árboles de decisión no siempre son la mejor herramienta para este tipo de análisis ya que el árbol de decisión de un sistema complejo con muchas secuencias de pasos y combinaciones de condiciones puede tener un tamaño considerable. El gran número de posibilidades con sus distintas alternativas puede suponer un problema más que una ayuda para el análisis. En estos casos los analistas corren el riesgo de no determinar qué políticas o estrategias de la empresa son la guía para la toma de decisiones específicas (en estos casos es aconsejable considerar las tablas de decisión).

Estructura

Un árbol de decisión consta de unas entradas, las cuales pueden ser un objeto o una situación descrita por medio de un conjunto de atributos, y a partir de ellas devuelve una respuesta. Los valores que pueden tomar las entradas y las salidas pueden ser valores discretos (clasificación) o continuos (regresión) aunque se utilizan más los valores discretos por simplicidad.

El árbol de decisión suele contener diferentes tipos de nodos: nodos internos, nodos de probabilidad, nodos hojas y ramas. Un nodo interno contiene un test sobre algún valor de alguna de las propiedades, un nodo de probabilidad indica que debe ocurrir un evento aleatorio de acuerdo a la naturaleza del problema, un nodo hoja representa el valor que devolverá el árbol de decisión y finalmente las ramas muestran los posibles caminos de los que se dispone de acuerdo a la decisión tomada. De esta forma, un árbol de decisión lleva a cabo un test a medida que recorremos el camino hacia las hojas para alcanzar así una decisión final. Los árboles de decisión pueden ser leídos como un conjunto de reglas, y a continuación mostraremos un pequeño ejemplo para comprender mejor su arquitectura.

Ejemplo

Nos encontramos ante un servicio de mensajería y queremos conocer las condiciones (color azul) por las cuales aplicaremos cada uno de los descuentos posibles (color rojo) en nuestro envío. Si observamos el árbol de la Figura 2.2, podemos ver que para aplicar un descuento del 50 % en el coste de nuestro envío, deberíamos solicitar un número de unidades superior a 50 con destino nacional.

De forma recíproca, si conocemos cada una de las condiciones de nuestro pedido, podemos saber el tipo de descuento que se nos aplicaría, por ejemplo: si solicitamos 25 unidades con destino europeo, se aplicaría un descuento del 10 %.

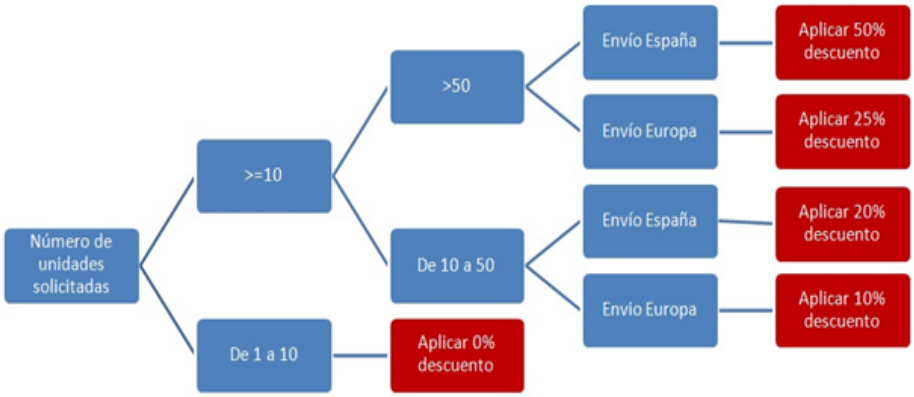


Figura 2.2: Ejemplo de un árbol de decisión

Este tipo de construcciones ayudan a las empresas a determinar cuáles son sus opciones así como los correspondientes resultados.

Algoritmos para la construcción de árboles de decisión

Existe una gran diversidad de algoritmos para la construcción de árboles de decisión. Entre los más destacados están:

- **ID3** (Induction Decision Trees): Se trata de un algoritmo propuesto por J. Ross Quinlan en 1986 [40] que recorre el árbol de decisión desde la raíz y tanto en ella como en cada uno de los demás nodos se decide qué rama tomar basándonos en el valor de algún atributo del objeto que se esté clasificando, hasta llegar a un nodo terminal (hoja), que corresponde a la clase final en a la cual pertenecerá dicho objeto.

- **C4.5**: Surge como una modificación propuesta por el propio Quinlan [41] con el fin de mejorar el algoritmo ID3. Construye el árbol de clasificación a partir del conjunto de entrenamiento y después de hacerlo utiliza una técnica de poda (*prunning*) basada en aplicar un test de hipótesis para deducir si merece la pena o no expandir dicha rama. El refinamiento que introduce respecto al ID3 es una medida alternativa llamada el ratio de ganancia, es decir, en cada nodo selecciona el atributo con mayor ratio de ganancia de información evitando así favorecer la elección de variables con un mayor número de valores. Una implementación de este algoritmo es la función J48 en la herramienta *Weka* [20] de minería de datos, y que será la que emplearemos en la práctica para construir los clasificadores.

- **CHAID** (Detector Automático de Chi-cuadrado de interacción): Se trata de un método [37] de clasificación jerárquico y descendente. Este algoritmo se basa en seleccionar la variable que más separa los grupos estudiados, usando para ello el test chi-cuadrado o la razón de verosimilitud. No aplica ninguna técnica de postpoda para evitar el sobrentrenamiento, sino que es el propio algoritmo el encargado de decidir cuándo se para de desarrollar el árbol. Sólo es capaz de tratar con variables predictoras discretas y su aplicación fundamental comenzó en el ámbito del diagnóstico médico y la detección de interacción entre variables para posteriormente ser usada en el ámbito del marketing.

Capítulo 3

Técnicas empleadas

En este capítulo vamos a profundizar en algunas de las técnicas que se han mencionado hasta ahora. Como hemos introducido previamente, muchas de ellas se basarán en la construcción de árboles de clasificación y en función del número de clasificadores que confeccione, podemos agrupar estos métodos en dos nuevas categorías: un clasificador o varios clasificadores. Respecto a los primeros, veremos algoritmos basados en la construcción de un único árbol de decisión, mientras que los segundos combinan varios clasificadores básicos con el fin de obtener uno más preciso (multiclasificadores). Finalmente, una vez conocido tanto el funcionamiento como las ventajas e inconvenientes de cada clasificador, propondremos algunos modelos que resultarán de combinar algunos de estos métodos con el objetivo de mejorar los resultados obtenidos por los mismos.

3.1. Tipos de clasificadores

Tal y como hemos mencionado, abordaremos algoritmos que se basan en la construcción de un sólo clasificador y otros que combinan varios clasificadores individuales. En el primer caso, utilizaremos técnicas que remuestrean los datos (resampling) para posteriormente construir el árbol de decisión que nos permitirá proceder con la clasificación. Se tratarán en profundidad las técnicas de *undersampling* [31], *oversampling* [24] y la combinación simultánea de ambos (algoritmos híbridos). En el segundo caso, nos centraremos en algunos de los multiclasificadores más utilizados hoy en día, como *Boosting* [2], *Bagging* [53] o *Random Forest* [30], que a pesar de no atacar de forma directa el problema del desbalanceo, pueden llegar a mejorar los resultados obtenidos por los algoritmos de resampling. Además, se han diseñado métodos que remuestrean los datos y que luego hacen uso de un multiclasificador como modelo predictor. En el presente trabajo abordaremos un ejemplo de este tipo y será el caso del método *AdaOUBoost* [38]. En la actualidad, existe una gran diversidad de modelos predictivos orientados a la clasificación de nuevos datos que pese a tener grandes aplicaciones prácticas e incluso proporcionar resultados que compitan con los obtenidos en este estudio, no se verán en este trabajo, como por ejemplo, la regresión logística [45].

3.1.1. Un clasificador

El objetivo es crear un modelo que prediga el valor de una variable de destino (variable respuesta) en función de diversas variables de entrada (variables explicativas). El árbol de decisión es una representación simple para clasificar nuevos ejemplos y el aprendizaje basado en este tipo de árboles son una de las técnicas más eficaces para la clasificación supervisada. Como ya hemos visto, los árboles de decisión utilizados en la minería de datos son de dos tipos fundamentalmente: árboles de clasificación y árboles de regresión. En este caso, trabajaremos con los primeros ya que se pretende predecir la clase a la que pertenecen los datos, y más en particular en esta sección, con aquellos que construirán solamente un árbol de clasificación.

Cuando nos encontramos ante el problema de desbalanceo de clases, una de las soluciones más intuitivas es la modificación de la distribución de los datos, en particular, incrementar el número de casos de la clase minoritaria (en nuestro caso, los clientes que se dan de baja en la compañía), para que el modelo sea capaz de detectarlas y que pueda predecir un número razonable de bajas.

Esta tarea se conoce como remuestreo (en inglés, *resampling*) que en el ámbito de la estadística se refiere a un conjunto de métodos que permiten realizar algunas de las siguientes operaciones:

1. Estimar la precisión de muestras estadísticas (medianas, varianzas y percentiles) mediante el uso de subconjuntos de datos disponibles o tomando datos de forma aleatoria de un conjunto dado.

2. Realizar test de permutación.

3. Validar modelos para el uso de subconjuntos aleatorios.

Para nuestro problema, utilizaremos el remuestreo con el fin de validar modelos sobre un subconjunto de datos, que será la primera etapa de lo métodos que a continuación presentaremos.

Undersampling

Consiste en balancear la distribución de los datos eliminando instancias de la clase mayoritaria, es decir, del grupo de clientes que no se dan de baja. A pesar de su sencillez y de la reducción del tiempo de procesado de los datos, existe el riesgo de eliminar elementos de la muestra potencialmente importantes en el proceso de clasificación, por eso se han desarrollado métodos capaces de realizar una selección inteligente sobre los elementos del conjunto de datos de la clase mayoritaria. La idea en la que se basan estos algoritmos es suponer que las instancias próximas entre sí tienen mayor probabilidad de pertenecer a la misma clase.

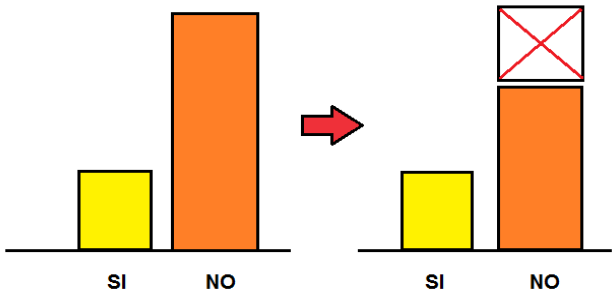


Figura 3.1: Undersampling

A continuación enunciamos varias de las técnicas de undersampling más utilizadas en la práctica:

- **Random undersampling**: suprimimos, al azar, algunas instancias de la clase mayoritaria.
- **Tomek Links**: [49] eliminamos solo instancias de la clase mayoritaria que sean redundantes o que estén muy próximos a instancias de la clase minoritaria.
- **Wilson Editing**: [50] también conocido como ENN (Editing Nearest Neighbour), elimina aquellas instancias donde la mayoría de sus vecinos pertenece a otra clase.

Además de éstas, se dispone de varias técnicas basadas en la eliminación de datos en la muestra que podemos consultar en [51] aunque el método de undersampling que emplearemos en los modelos descritos en este trabajo es el primero, *random undersampling*, por su rápida y sencilla aplicación a nivel computacional. Los pasos que seguiremos para la aplicación del modelo se resumen a continuación, y su estructura se refleja en el esquema que se muestra en la Figura 3.2:

1. Dividimos la muestra de entrenamiento (en inglés, muestra training) en dos subconjuntos: uno con los clientes que no permiten acciones comerciales sobre sus cuentas, y otro con los que sí permiten estas acciones[3].
2. Sobre la muestra de clientes que no permiten acciones comerciales sobre sus cuentas, separamos los que sí se dan de baja de los que no.
3. Aplicamos la técnica de undersampling para reducir el número de casos de la clase mayoritaria (clientes que no se dan de baja).
4. Concatenamos la nueva muestra generada con los casos de la clase minoritaria (clientes que sí se dan de baja).
5. Aplicamos el modelo sobre esta muestra de datos.

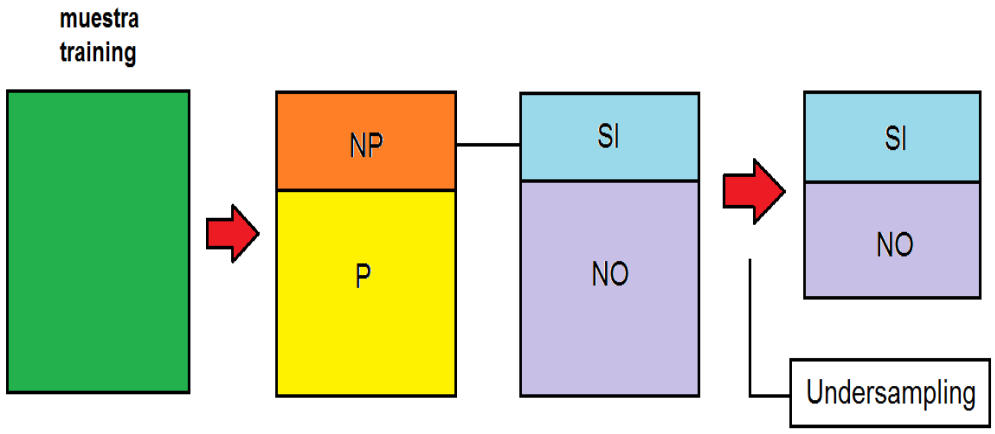


Figura 3.2: Estructura de la técnica de undersampling

Para la aplicación del modelo sobre la muestra de datos que construimos después de aplicar la técnica de undersampling, utilizaremos la función *J48* [52] de la librería *RWeka* de R, con el que construiremos el árbol de decisión requerido para la clasificación de los datos y que se puede consultar en el apéndice de este trabajo.

[3] *En este trabajo entrenaremos los clasificadores sobre aquellos clientes que no permiten acciones comerciales sobre sus cuentas ya que se trata de datos estáticos, es decir, permanecen constantes durante la ejecución del modelo. Esta partición de la muestra se aplicará como primer paso en todos y cada uno de los métodos que en este trabajo revisaremos.*

Oversampling

Consiste en balancear la distribución de los datos añadiendo ejemplos de la clase minoritaria, es decir, del grupo de clientes que sí se dan de baja. Además de aumentar el tiempo de procesado de los datos, el principal problema de este método tiene lugar cuando generamos ejemplos ruidosos que se producen al realizar un elevado número de réplicas de ciertas instancias, modificando en exceso la distribución de las bajas, y que puede derivar en un incremento desmesurado de las probabilidades de baja finales.

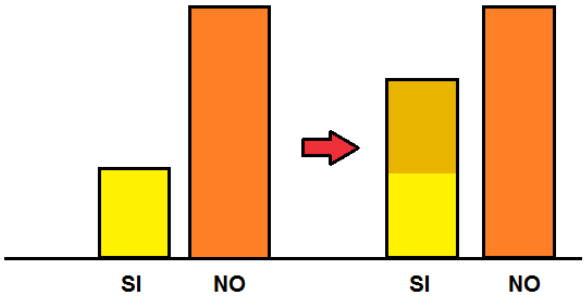


Figura 3.3: Oversampling

Al igual que en el método de undersampling, esta adición puede realizarse de varias formas y a continuación mencionamos algunas de las más destacadas:

- **Random oversampling**: realizamos réplicas exactas de las ya existentes en la muestra, lo que puede producir el fenómeno de *overfitting*. En concreto, realizaremos copias enteras (duplicaciones, triplicaciones, etc.) de los casos de la clase minoritaria para garantizar así la igualdad de los registros.

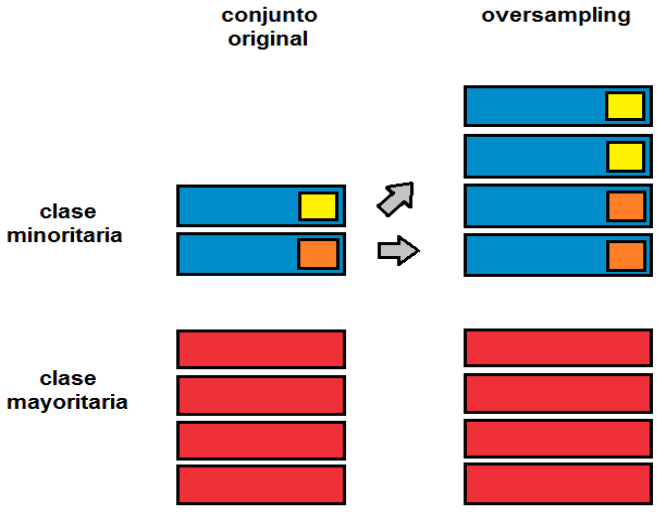


Figura 3.4: Duplicación de la clase minoritaria

- **SMOTE (Synthetic Minority Oversampling Method)**: Esta técnica [11] genera nuevas instancias de la clase minoritaria interpolando los valores de las instancias minoritarias más cercanas a una dada.

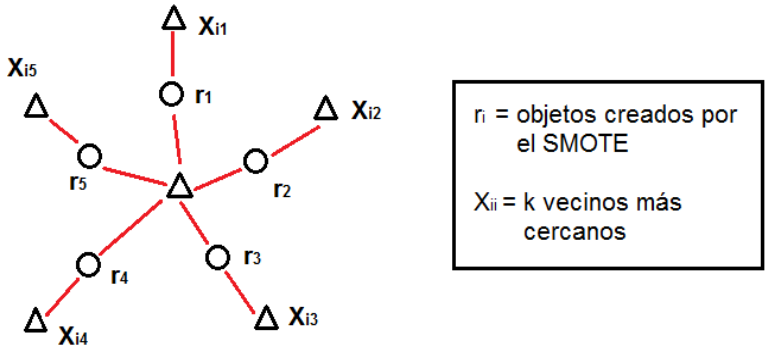


Figura 3.5: Synthetic Minority Oversampling Technique.

En [36] se describe con detalle el funcionamiento de esta técnica de undersampling, en particular, una versión de la misma conocida como SMOTE-I, que mejora los resultados obtenidos por el SMOTE y que, a diferencia de éste, trabaja con más de una clase minoritaria.

En la práctica emplearemos ambas técnicas aunque, para este primer caso de uso, hemos utilizado el primero de ellos, random oversampling. El proceso a seguir sería similar al utilizado en el algoritmo de undersampling salvo algunos matices:

1. Dividimos la muestra de entrenamiento en dos subconjuntos: uno con los clientes que no permiten acciones comerciales sobre sus cuentas, y otro con los que sí permiten estas acciones.
2. Sobre la muestra de clientes que no permiten acciones comerciales sobre sus cuentas, separamos los que sí se dan de baja de los que no.
3. Aplicamos la técnica de oversampling para incrementar el número de casos de la clase minoritaria (clientes que sí se dan de baja).
4. Concatenamos la nueva muestra generada con los casos de la clase mayoritaria (clientes que no se dan de baja).
5. Aplicamos el modelo sobre esta muestra de datos.

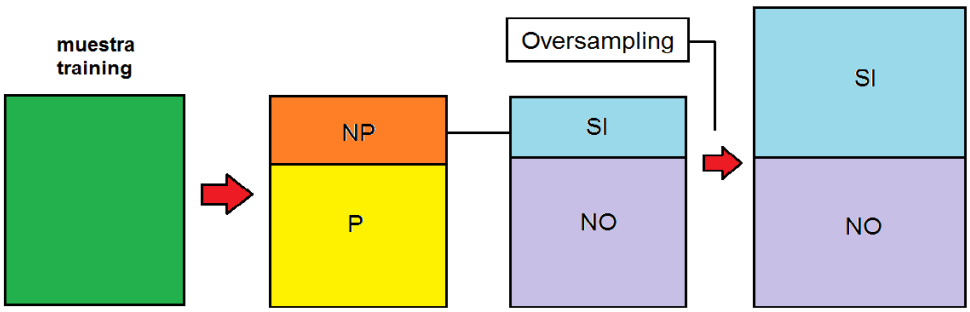


Figura 3.6: Estructura de la técnica de oversampling

Para la construcción del modelo, utilizaremos de nuevo la función *J*48 de la librería *RWeka* de la que dispone R y que podemos consultar en el apéndice de este trabajo.

Algorimtos híbridos

Consisten en combinar las técnicas de undersampling y oversampling que acabamos de describir. Estos algoritmos constituyen una de las técnicas más empleadas hoy en día por los investigadores en cuanto a la corrección del problema del desbalanceo ya que se compone de dos técnicas que corrigen este problema modificando la distribución de los datos, además de tener una simple implementación a nivel computacional y que luego detallaremos. A pesar de ello, sigue conservando los inconvenientes que lleva implícitos en cada una de las dos técnicas que lo conforman (undersampling y oversampling).

Existen varios métodos de hibridación como consecuencia de la gran diversidad de algoritmos de undersampling y oversampling que se han ido desarrollando, sin embargo en este trabajo combinaremos random undersampling con random oversampling, dada su practicidad. Los pasos que hemos de seguir para aplicar esta técnica a nuestro modelo predictivo son los siguientes:

1. Dividimos la muestra de entrenamiento en dos subconjuntos: uno con los clientes que no permiten acciones comerciales sobre sus cuentas, y otro con los que sí permiten este tipo de acciones.
2. Sobre la muestra de clientes que no permiten acciones comerciales sobre sus cuentas, separamos los que sí se dan de baja de los que no.
3. Aplicamos la técnica de oversampling para incrementar el número de casos de la clase minoritaria (clientes que sí se dan de baja).
4. Aplicamos la técnica de undersampling para reducir el número de casos de la clase mayoritaria (clientes que no se dan de baja).
5. Concatenamos las dos clases en una sola muestra.
6. Aplicamos el modelo sobre esta muestra de datos.

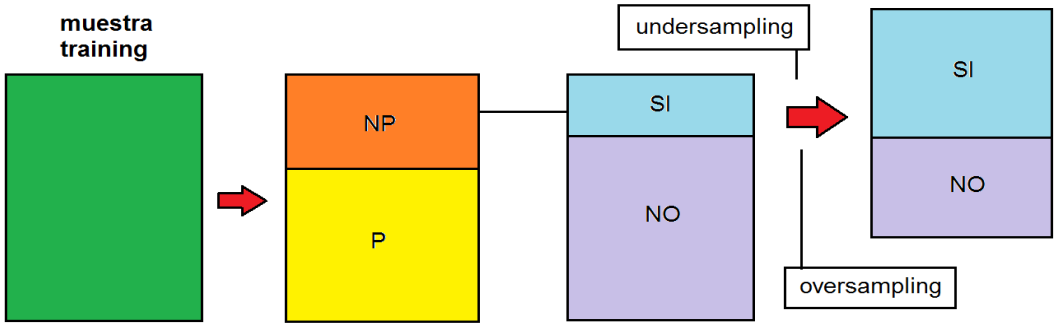


Figura 3.7: Estructura de los algoritmos híbridos.

Para la aplicación del modelo utilizaremos de nuevo la función *J48* de la librería *RWeka* y que podemos consultar en el apéndice de este trabajo. En el cuarto capítulo podremos consultar todos los resultados obtenidos en función del grado de modificación de la muestra con estos algoritmos de resampling.

3.1.2. Multiclasificadores

A pesar de los diversos estudios realizados hasta la actualidad en relación a los clasificadores, no existe todavía uno por excelencia que destaque sobre los demás, por lo que resulta complicado seleccionar un clasificador que logre encontrar una mejor frontera de decisión para discernir entre las clases. Sin embargo, en la búsqueda de mejores métodos de clasificación aparece cierta tendencia a combinar varios clasificadores en el mismo problema. En esto último se basan los algoritmos de multiclasificación, ya que combinan las diferentes salidas proporcionadas por los clasificadores con el objetivo de alcanzar un mejor resultado. Un sistema multiclasificador puede llegar a ser mejor que un clasificador simple [27], ya que algunos algoritmos ejecutan búsquedas que pueden llevar a diferentes óptimos de carácter local. La precisión obtenida al combinar distintos clasificadores supera, generalmente, la precisión obtenida por cada componente en particular y este hecho lo veremos más adelante en la práctica. La gran ventaja de los multiclasificadores reside en la corrección del problema del sobreajuste (*overfitting*) de los datos de entrenamiento. Además, los dos puntos clave que afectan a la calidad de un modelo multiclasificador son principalmente la construcción de modelos suficientemente diferentes y precisos, así como la forma óptima de combinarlos. A pesar de ello, los modelos generados fruto de esta síntesis de subclasificadores los convierte en modelos de difícil comprensión. A continuación, desarrollamos algunos de los más importantes en la práctica.

Boosting

Este multiclasificador nace en el año 1989 de la mano del profesor Robert Schapire aunque un año después es mejorado por John E. Freund. Se basa en la construcción de sucesivos clasificadores, sobre modificaciones de la muestra de entrenamiento realizadas en función de los errores cometidos por el clasificador en cada una de las iteraciones, que posteriormente combinará para obtener el clasificador final. Esto evidencia una clara dependencia entre los subclasificadores, lo cual constituye uno de los inconvenientes más grandes de este método. El Adaboost [17] es el algoritmo boosting más utilizado ya que dirige su atención en aquellos casos que son más difíciles de clasificar. El procedimiento en el que se basa este algoritmo multiclasificador para construir el modelo de clasificación es el siguiente:

1. Inicialmente a todos los datos del conjunto de entrenamiento (muestra training) se les asigna el mismo peso, $w_i = \frac{1}{n}$, donde n es el tamaño de la muestra.
2. Se procede con una primera clasificación y se calcula el error cometido por el modelo usando la muestra training, contando e identificando los objetos mal clasificados.
3. Se incrementan los pesos de los casos de la muestra training que el modelo ha clasificado de forma errónea.
4. Entrenamos de nuevo el modelo usando el conjunto de pesos modificados.
5. Volvemos al punto 2 y repetimos este procesos tantas veces como el número de iteraciones que se haya fijado.
6. Nuestro modelo final será una votación ponderada por los pesos de todos los modelos que hemos construidos durante la ejecución del algoritmo.

Para comprender un poco mejor el funcionamiento del método Adaboost vamos a suponer que construimos 3 subclasificadores, es decir, el número de iteraciones en el proceso de clasificación serán tres. En este caso, podríamos resumir el boosting con el esquema que en la Figura 3.8 se muestra.

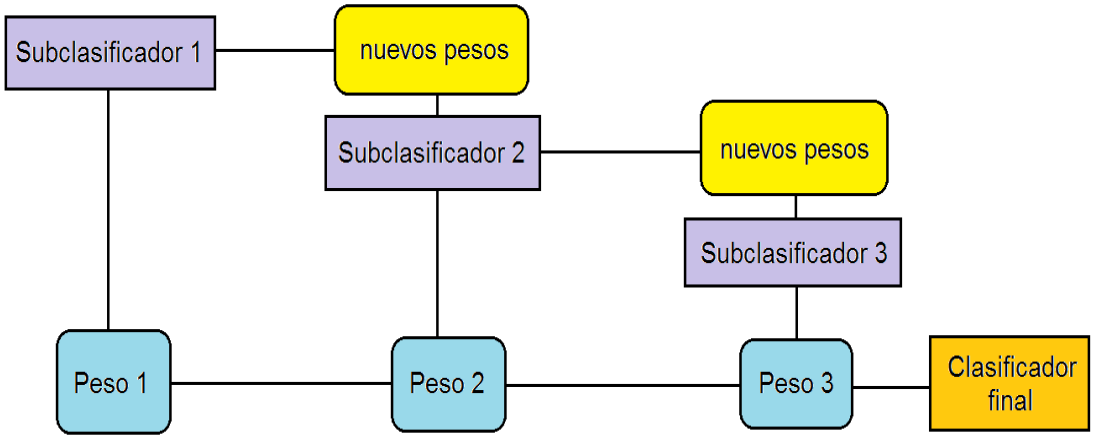
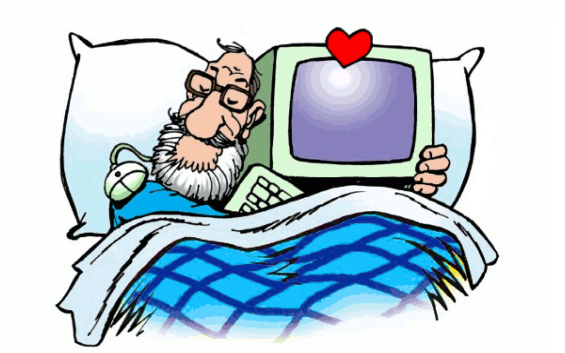


Figura 3.8: Estructura del método boosting

Además, su implementación es sencilla y puede hacerse de forma directa a través de la función *boosting* de la librería *adabag* en donde debemos establecer el número de iteraciones que queramos hacer. El código que se ha desarrollado para la implementación del algoritmo boosting puede consultarse en el apéndice de este trabajo.

¿Cómo podemos decidir el número de iteraciones que se deben realizar?

Cabe pensar que, al existir una fuerte dependencia entre los clasificadores, cuanto mayor sea el número de iteraciones empleado, mayor será el tiempo necesario para construir el modelo de predicción ya que no es posible realizar una construcción en paralelo de los subclasificadores. Tras las diferentes pruebas realizadas destacaremos que para un número inferior a 5 iteraciones, el método boosting constituye un algoritmo ágil para la clasificación de nuestros datos. Sin embargo, a partir de este límite, el algoritmo iba perdiendo rapidez ya que los tiempos de procesado llegaban a superar la hora de ejecución, lo cual lo convierte en un modelo poco práctico.



Tras los recientes estudios, se ha establecido un test de parada que especifica que cuando el error de clasificación cometido sea mayor o igual que 1/2, entonces se finaliza la ejecución. Más adelante, realizaremos un estudio comparativo de los diferentes resultados obtenidos a partir de varios ensayos sobre el número de subclasificadores a generar para la confección del modelo boosting (tercera sección del capítulo cuarto del presente trabajo).

Bagging

El método Bagging (**B**ootstrap **Agg**regat**ing**) [3] es un clasificador estadístico propuesto por Leo Breiman en el año 1994. En primer lugar, genera a partir de la muestra inicial varios subconjuntos del mismo tamaño con reemplazamiento (asegurando así la diversidad) y de forma independiente. A continuación, para cada uno de ellos, construye un subclasificador y, finalmente, utilizando el voto mayoritario obtiene la clasificación final para cada individuo. A diferencia del boosting, en este método no existe ningún tipo de dependencia entre los subclasificadores por lo que el tiempo necesario para construir el clasificador final no aumenta de forma proporcional al número de subclasificadores utilizados. Esta forma de proceder puede verse de forma más gráfica en la Figura 3.9 y en la que consideraremos el caso general de n subclasificadores:

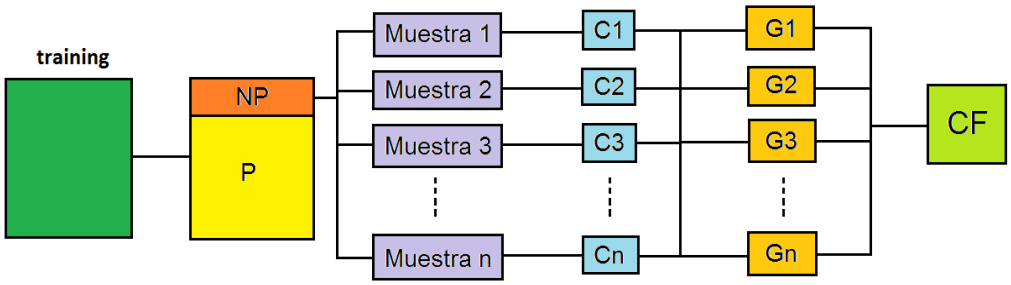


Figura 3.9: Estructura del método bagging

Su implementación en el lenguaje de programación R es muy sencilla (consultar apéndice) y puede hacerse a través de la función *bagging* de la librería *adabag* en donde indicaremos el número de árboles que queremos construir para obtener el clasificador final.

¿Cuántos subclasificadores construimos?

El método bagging es capaz de construir todos los subclasificadores de forma simultánea, lo que supone un gran ahorro en el tiempo de procesado del modelo. Sin embargo, la función *bagging* de la librería *adabag* lo hace paso a paso ya que cuanto mayor sea el número de árboles empleado, más se tarda en confeccionar el clasificador final. A pesar de ello, existen algoritmos que permiten realizar esta clasificación en paralelo. En nuestro caso, hemos realizado un estudio comparativo que nos permita decidir sobre esta cuestión y que en el cuarto capítulo de este trabajo discutiremos apoyándonos en los resultados obtenidos para cada modelo.

Bagging vs Boosting

Hasta ahora hemos visto dos algoritmos diferentes en el campo de la multiclasificación: boosting y bagging. Ambos métodos tienen similitudes, por ejemplo, ambos construyen los clasificadores básicos sobre versiones modificadas del conjunto de entrenamiento y los combinan después en la regla de clasificación final. Sin embargo, se diferencian en la forma de obtener los subclasificadores. En particular, en el boosting utilizamos todas las observaciones en cada iteración (en el bagging no necesariamente) y existe una dependencia en los subclasificadores construidos por el boosting mientras que estos se generan de forma independiente en el bagging. Para datos con bajo nivel de ruido, boosting tiene mejor rendimiento que bagging, sin embargo, si los datos de entrenamiento contienen ejemplos ruidosos, entonces boosting degrada su rendimiento ya que focaliza el algoritmo en este tipo de datos.

Random Forest

En el año 2001 Leo Breiman [5] propone este nuevo multiclasificador que consiste en un refinamiento del método bagging que acabamos de mostrar. Presenta la misma estructura que él a excepción de una previa selección de variables antes de generar las muestras bootstrap (Figura 3.10). Esto le permite ser un clasificador muy certero aunque de compleja interpretación. Trabaja de forma eficiente con bases de datos grandes y puede manejar cientos de variables sin excluir a ninguna. Sin embargo, en los casos en los que la base de datos contiene variables categóricas con diferente número de niveles, el Random Forest se parcializa a favor de aquellos atributos con más categorías. Por tanto, la posición que marca la variable no es fiable para este tipo de datos aunque existen soluciones que remedian esta cuestión, por ejemplo, las permutaciones parciales. La forma de construir el clasificador final mediante este método puede resumirse en los pasos que a continuación resumimos y que obedecen a la arquitectura de la Figura 3.10.

1. Creamos aleatoriamente varios conjuntos de entrenamiento (con reemplazamiento) con el mismo tamaño que el conjunto original (muestra training). Al seleccionarse de esta forma, no todos los datos de la muestra training estarán necesariamente en los subconjuntos de entrenamiento. En cada punto de división del árbol, la búsqueda de la mejor variable para dividir los datos no se realiza sobre todas las variables de las que dispone el modelo, sino sobre un subconjunto de las mismas. Se busca la mejor división de los datos de entrenamiento teniendo en cuenta solo las variables seleccionadas.

2. Para cada uno de los diferentes subconjuntos construidos se obtiene el correspondiente subclasificador.

3. Una vez construidos todos los subclasificadores, la evaluación de cada nueva entrada se realiza con el conjunto de árboles generados por el Random Forest. La categoría final de cada individuo (clasificación) se lleva a cabo mediante el voto mayoritario del conjunto de árboles, y en caso de regresión, por el valor promedio de los resultados.

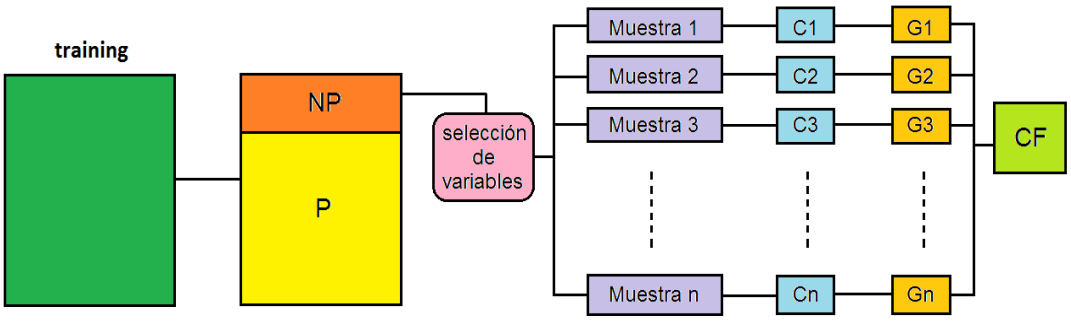


Figura 3.10: Estructura del método Random Forest

Su implementación es sencilla, directa y se puede realizar a través de la función *randomForest* de la propia librería *randomforest* de R, en donde tendremos que especificar el número de árboles que queremos construir así como el número de variables empleadas para cada una de ellos. Todo esto puede consultarse en el apéndice de este trabajo.

¿Cuántas variables podemos seleccionar?

Intuitivamente cabe pensar en que debemos elegir aquellas variables que mejor puedan entender nuestras áreas de negocio. La ventaja de la función *randomForest* es que imprime por pantalla una lista con la importancia de cada una de las variables presentes en el modelo (salida *importance* del modelo), lo cual nos puede ayudar a decidirnos sobre qué variables elegir para construir el clasificador. Además, existe la posibilidad de visualizarlo a través de la función *varImpPlot* que muestra un gráfico con los valores del *importance* correspondientes a cada una de las variables empleadas por el modelo. A continuación se muestra un ejemplo en donde emplearemos la base de datos *mtcars* extraídas de la revista americana *Motor Trend* del año 1974 y que contiene información sobre el consumo de combustible de 32 automóviles diferentes. En particular, se cuentan con un total de 11 variables, en donde nuestra variable respuesta serán las millas por galón (mpg) frente a las demás variables que serían las explicativas.

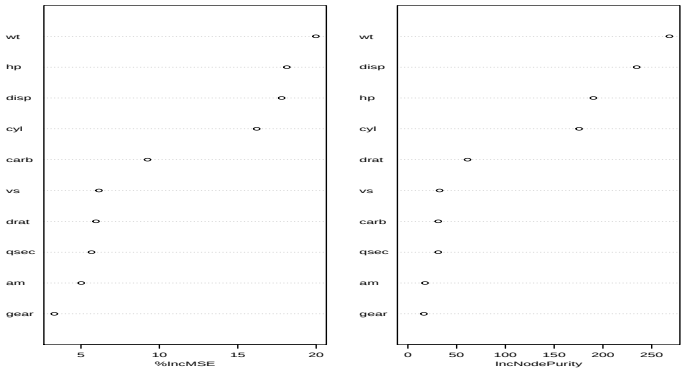


Figura 3.11: Importancia de las variables

Como se puede observar, las variables que el modelo considera como más relevantes para explicar la respuesta son el peso (wt), seguido del desplazamiento (disp) y la potencia (hp) del vehículo. Estos datos se pueden obtener de forma explícita a través de la función *importance* y que a continuación podemos comprobar:

```
> importance(mtcars.rf)
```

	%IncMSE	%IncNodePurity
cyl	16.306732	161.41242
disp	18.334288	258.19116
hp	18.688350	185.43380
drat	7.032319	64.58365
wt	19.206956	254.75402
qsec	4.209702	30.77328
vs	5.797489	27.70832
am	3.861202	11.17482
gear	4.752359	17.99918
carb	7.751532	28.97506

Esta constituye una herramienta útil para decidir qué y cuántas variables debemos elegir para construir nuestro modelo. Sin embargo, existen otras alternativas.

En general, se suele utilizar la raíz cuadrada del número total de variables presentes en los datos. En nuestro caso, contamos con un total de 72 variables, por tanto, escogeremos

$$\sqrt{72} = 8{,}4853 \approx 8 \text{ variables.}$$

Para ver como varían los resultados en función del número de variables elegidas contamos con un estudio comparativo que se mostrará en el capítulo cuatro de este trabajo.

¿Cómo seleccionamos el número de árboles?

Recordemos que la construcción de cada uno de los árboles se realiza en paralelo, por tanto, el tiempo de procesado no aumenta en proporción al número de árboles empleados. Para hacer una correcta selección, podemos realizar una gráfica de los errores de clasificación cometidos por cada modelo y escoger el número de árboles a partir del cual este error se estabiliza. En la Figura 3.12 se muestra un ejemplo en el que se realiza un gráfico de los errores de clasificación cometidos por el modelo utilizando desde 1 hasta 100 árboles. En este gráfico habrá tantas líneas como cantidad de clases se quiera predecir (en nuestro caso dos: baja y no baja) además del error en la clasificación correspondiente a los datos no presentes en cada submuestra. Cuando todas las líneas se superponen, entonces todas las medidas tienen el mismo error y no tiene sentido identificarlas. Podemos observar que el error oscila levemente hasta estabilizarse a partir de los 10 árboles, que sería el número de clasificadores que bastaría construir para obtener el clasificador final.

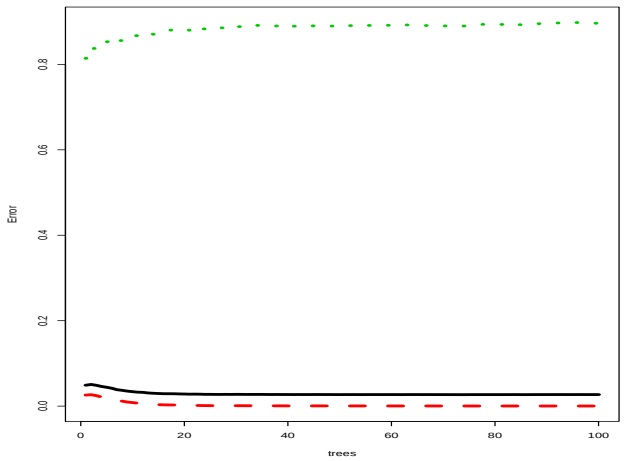


Figura 3.12: En el eje X se recogen el número de árboles utilizados en cada caso y en el eje Y el error en la clasificación cometido. La línea de color verde muestra el valor medio del error obtenido para la clase minoritaria mientras que la línea roja corresponde al error medio de la clase mayoritaria. La línea negra corresponde al error medio de los casos no presentes en los subconjuntos de datos creados a partir de la muestra original (OOB).

AdaOUBoost

Los algoritmos de multiclasificación que hemos visto hasta ahora no solucionaban el problema del desbalanceo. Por tanto, seguidamente propondremos un nuevo clasificador que sí tiene en cuenta esta cuestión y además sigue conservando la idea de combinar distintos subclasificadores para obtener uno más potente (multiclasificador). Nos referimos al método AdaOUBoost (Adaptative Oversampling-Undersampling Boosting) [38] y que, en primer lugar, modifica la distribución de los datos a través de las técnicas de undersampling y oversampling que hemos visto antes, para después construir un clasificador final más robusto utilizando el método boosting. Debido a la gran diversidad de métodos de resampling, existen muchas formas de implementar este método, sin embargo, explicaremos a continuación la metodología que hemos utilizado en este trabajo para llevarlo a la práctica:

1. Aplicamos en primer lugar la técnica de undersampling utilizando un método clúster para dividir la clase mayoritaria en varios subconjuntos disjuntos (no necesariamente del mismo tamaño).
2. Utilizamos la técnica borderline-SMOTE [23] para incrementar los casos de la clase minoritaria (lo cual se correspondería con la parte de oversampling).
3. Concatenamos la clase minoritaria con cada uno de los subconjuntos de la clase mayoritaria que hemos dividido.
4. Construimos un clasificador para cada uno de estos conjuntos utilizando la función J48 de la librería *RWeka*.
5. Promediamos los valores de las predicciones obtenidas por cada subclasificador para obtener la predicción del árbol final y aplicamos la siguiente regla:
   - Si la probabilidad es mayor que 0.5, le asignamos un SI, es decir, predecimos una baja para ese cliente.
   - Si la probabilidad es menor que 0.5, le asignamos un NO, es decir, no predecimos una bajara para ese cliente.

Para comprender mejor el funcionamiento de esta técnica, vamos a mostrar un ejemplo de la estructura que adoptaría para un número determinado de subclasificadores. Supongamos que dividimos la clase mayoritaria en tres subconjuntos después de aplicar la clusterización. En este caso, obtendríamos el esquema mostrado en la Figura 3.13.

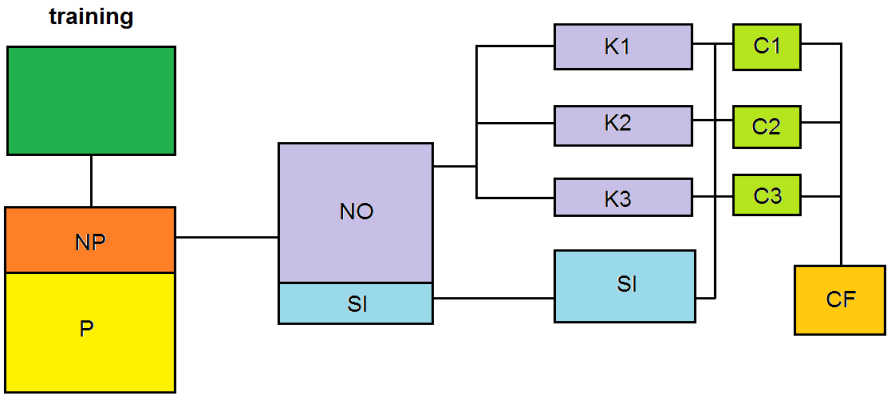


Figura 3.13: Estructura del método AdaOUBoost

Hemos de mencionar que en la práctica, no hemos aplicado la técnica borderline-smote como tal, sino que hemos utilizado la técnica SMOTE que antes hemos visto y que remuestrea ambas clases al mismo tiempo y de la siguiente forma:

1. Se genera un número determinado de casos de la clase minoritaria.
2. Por cada caso de la clase minoritaria generado, eliminamos un número determinado de casos de la clase mayoritaria.

Esto puede implementarse de forma directa y sencilla a través de la función *smote* de la librería *DmWR* de la que dispone R (consultar apéndice del trabajo).

¿Número de clusters?

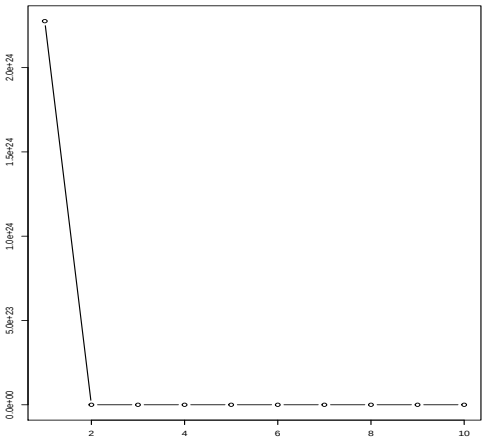


Figura 3.14: Número de clusters: en el eje de abscisas contamos con la cantidad de clusters mientras que el eje de ordenadas alberga la suma de errores cometidos en cada caso.

Uno de los mayores inconvenientes de este método es una incorrecta selección del número de grupos (clusters) en los que dividimos inicialmente la muestra correspondiente a la clase mayoritaria. Una mala elección del número de clusters puede ser causada por varias de la siguientes razones:

- Si dos centroides iniciales caen por casualidad en el mismo clúster, entonces puede que los demás grupos estén poco diferenciados entre sí.
- Si hay algún dato atípico, se obtiene por lo menos un cluster con objetos muy dispersos.

Para hacer una correcta partición existen varios métodos, por ejemplo, podemos fijar un número k de grupos inicialmente. Para cada uno de estos clusters, se realiza un gráfico con los errores cometidos al emplear cada uno de ellos, y en donde veamos un "codo" o un pico pronunciado , su correspondiente valor de k será el número de grupos utilizados para la división de la clase mayoritaria.

Atendiendo al ejemplo de la Figura 3.14, nos decantaríamos por la cantidad de dos grupos ya que a partir de ese grupo el error se normaliza. Por otro lado, podemos construir una función de simulación que ensaye distintos valores para este k y vaya mostrando los resultados obtenidos para cada uno, y de esta forma poder compararlos para decidir sobre el mejor valor que podemos asignarle a esta variable.

En este trabajo se han abordado ambas metodologías ya que hemos considerado este paso como fundamental para el buen desarrollo del algoritmo. Su implementación puede consultarse el apéndice de este trabajo.

3.2. Criterios de selección

Son muchas las técnicas que se han ido revisando durante los últimos años para poder hacer frente al problema de desbalanceo de clases en los modelos predictivos y en este trabajo no se han revisado en su totalidad. Sin embargo, nuestra selección de los clasificadores no ha sido fruto del azar, sino que existen varias razones que justifican esta elección. En la siguiente tabla recogemos algunas conclusiones sobre cada uno de los métodos que hemos descrito hasta ahora y que nos permiten razonar los motivos por los cuales han sido seleccionados para este trabajo.

Técnica	**Ventajas**	**Inconvenientes**
Undersampling	Menor tiempo de procesado	Pérdida de información
Oversampling	No hay pérdida de información	Repetición de muestras ruidosas y mayor tiempo de procesado
AdaOUBoost	Multiclasificador que aborda el problema del desbalanceo	Elección inapropiada del número de clusters
Boosting	Implementación computacional sencilla.	Overfitting y dependencia entre los subclasificadores Poco práctico para un número elevado de iteraciones.
Bagging	Bueno para modelos inestables y reduce el sobreentrenamiento de los modelos.	Puede perjudicar procesos estables y, en ocasiones, poco práctico a nivel computacional
Random Forest	Produce un clasificador muy certero	Interpretación compleja y riesgo de hacer una incorrecta selección del número de árboles.

Tabla 3.2. Ventajas e Inconvenientes de las técnicas de desbalanceo

La elección de estas técnicas están estrictamente ligadas a los objetivos marcados por el proyecto. Entre otras cuestiones, nos referimos a la rápida ejecución de los métodos así como su sencilla implementación con el lenguaje *R* y la obtención de unos resultados fácilmente interpretables. Por otro lado, desde el punto de vista de los modelos predictivos, se pretendía obtener clasificadores que proporcionasen unos resultados fiables. Este último aspecto puede comprobarse en el siguiente capítulo, en donde recogemos los resultados proporcionados por cada una de las técnicas empleadas en este trabajo.

Capítulo 4

Aplicación a datos reales

En este capítulo aplicaremos todas las técnicas que se han visto hasta ahora sobre un caso real, el número de bajas de clientes en una compañía telefónica determinada. Empezaremos por ubicar el escenario en el que nos encontraremos y que nos permita identificar en qué punto se detecta el problema del desbalanceo de clases para después establecer las herramientas que utilizaremos para tratarlo. A continuación, desarrollaremos un proceso de evaluación que nos permita cuantificar el buen hacer de todas las técnicas que hemos descrito en este trabajo para entender cómo podremos evaluarlas una vez que se hayan desplegado y aplicado a nuestros datos. Finalmente, se ofrecerán los resultados que se obtendrían a través de las mismas (y las cuestiones que han ido sugiriendo: número de iteraciones, número de árboles a construir etc.) así como una serie de modelos que hemos propuesto como complemento al estudio realizado.

4.1. Escenario

Para este proyecto, se dispuso de una serie de ficheros, convenientemente anonimizados, con información acerca de las cuentas, opciones, incidencias, reclamaciones, etc., que hemos utilizado para la construcción de los modelos predictivos sobre los cuales aplicaremos las distintas técnicas de desbalanceo que en el capítulo 3 hemos desarrollado. Estos datos corresponden al trimestre formado por los meses de Enero, Febrero y Marzo del año 2014.

Para comprender mejor la aplicación de los métodos que hemos descrito en el apartado anterior, vamos a recurrir a una herramienta gráfica llamada KNIME que nos permite visualizar cada una de las etapas en las que se divide el problema y que en la Figura 4.1 mostramos de forma esquemática. En la siguiente sección entraremos en más detalles con esta herramienta y aunque en la práctica no ha sido la que hemos utilizado en este trabajo, constituye una manera muy cómoda de visualizar las diferentes fases en la que dividimos nuestro problema, desde el tratamiento de la base de datos hasta la validación de los modelos de clasificación.

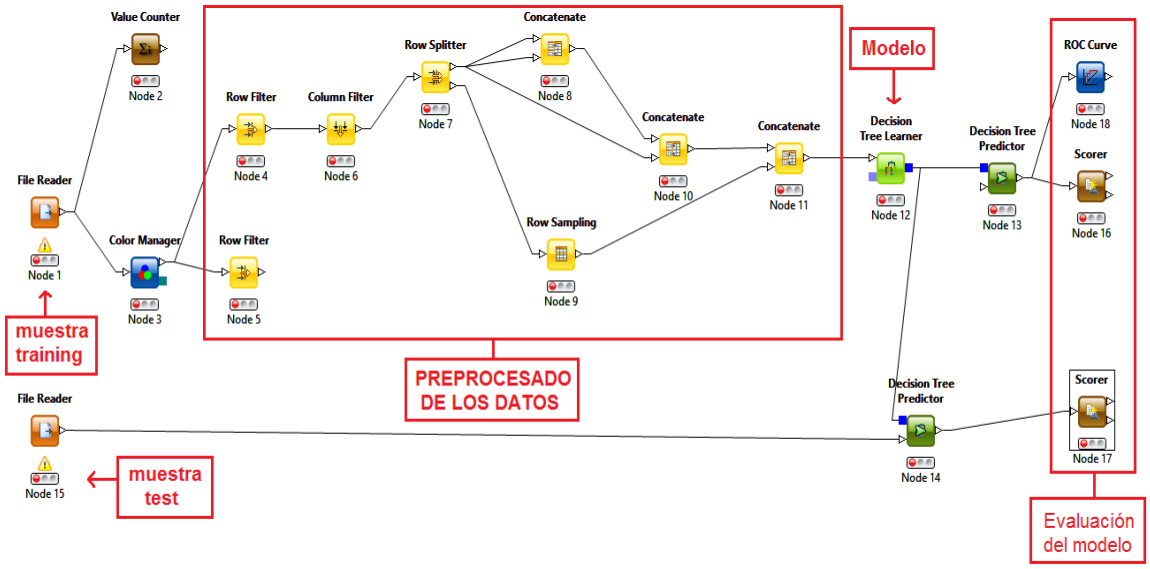


Figura 4.1. Escenario del problema en KNIME.

Como podemos observar, se aprecia una primera etapa en la que pre-procesaremos los datos aunque solo se producirá en aquellos algoritmos que requieran un modificación en la distribución de la muestra (undersampling, oversampling...). A continuación, se lleva a cabo la construcción del modelo a través de los árboles de decisión así como las predicciones obtenidas por el mismo. Finalmente, se incurre en el proceso de evaluación del modelo utilizando las métricas que más adelante explicaremos además de otros criterios más gráficos para la comparación de modelos de clasificación como la curva ROC.

4.1.1. Herramientas de analítica

Para la parte práctica de este proyecto se han utilizado dos herramientas de trabajo: R y KNIME. Aunque todo el código que en el anexo del trabajo se incluye está desarrollado con R, KNIME ha sido más útil en cuanto a aspectos visuales del problema como el ya mencionado escenario del problema.

R

R es un entorno y lenguaje de programación con un enfoque al análisis estadístico, además de ser una implementación de software libre del lenguaje S pero con soporte de alcance estático. Se trata de uno de los lenguajes más utilizados en investigación por la comunidad estadística, siendo muy popular en el campo de la minería de datos, la investigación biomédica, la bioinformática y las matemáticas financieras. A esto contribuye la posibilidad de cargar diferentes bibliotecas o paquetes con funcionalidades de cálculo o graficación.

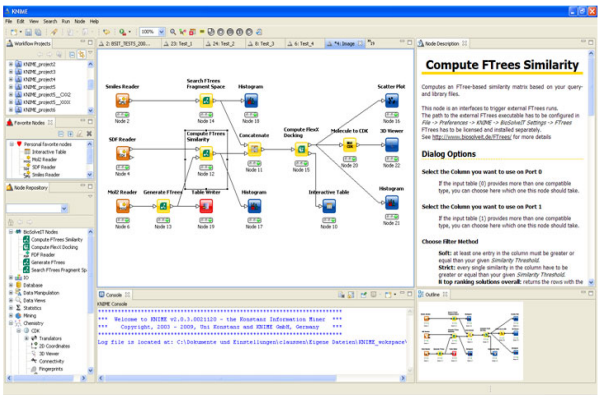
Fue desarrollado inicialmente por Robert Gentleman y Ross Ihaka del Departamento de Estadística de la Universidad de Auckland en 1993.

El desarrollo de proyectos en R se basa en un conjunto de comandos que pueden previamente desarrollarse en cualquier editor. En la actualidad también existe algún IDE (Integrated Development Environment) que permite realizar el desarrollo de una forma más amigable, siendo el principal exponente RStudio, el cual se ha empleado para la elaboración del código que en el anexo de este trabajo se muestra.

KNIME

KNIME (Konstanz Information Miner)[5] es una herramienta ampliamente aplicada a minería de datos que permite el desarrollo y ejecución de técnicas mediante modelos en un entorno visual, utilizada por más de 3.000 organizadores.

Fue desarrollado originalmente en el departamento de bioinformática y minería de datos de la Universidad de Constanza, Alemania, bajo la supervisión del profesor Michael Berthold. Se distribuye bajo la Licencia Pública General de GNU (GNU General Public License) que es la licencia más usada en el mundo del software y garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir y modificar el software.



Está concebido como una herramienta gráfica y dispone de una serie de nodos (que ofrecen distintos tipos de algoritmos) y flechas (que representan flujos de datos) que se despliegan y combinan de manera gráfica e interactiva.

La interfaz gráfica de usuario permite el montaje fácil y rápido de nodos para el preprocesamiento de datos, extracción, transformación y carga (ETL), modelado, análisis de datos, aprendizaje automático, visualización y minería de datos a través de su concepto modular de canalización de datos (data pipelining).

KNIME no sólo se utiliza en la investigación farmacéutica, también se usa para análisis de datos de cliente de CRM, en inteligencia de negocio y para análisis de datos financieros.

---

[5] *https://www.knime.org/*

4.2. Evaluación del modelo

Para medir el rendimiento de un modelo clasificador debemos fijarnos en la tasa de error cometido por el mismo, es decir, tendremos que calcular el porcentaje de casos clasificados de forma incorrecta sobre el conjunto de datos con los que estamos trabajando.

Ciertamente, la tasa de error en el conjunto de datos sobre el que se ha generado el modelo es un mal predictor del rendimiento del mismo, ya que puede ocurrir el fenómeno de *overfitting* que sucede cuando el modelo tiene una tasa de error muy pequeña. Esto no significa que el modelo sea muy bueno haciendo nuevas predicciones ya que, realmente, ha aprendido "de memoria" el patrón que siguen los datos. Por tanto, si medimos el error de predicción sobre los mismos datos con los que se ha generado el modelo, éste seguramente sea muy pequeño pero cuando se hacen predicciones sobre otros datos el error puede llegar a dispararse.

Para evitar este problema, una posible solución es dividir los datos en dos subconjuntos distintos: la muestra de entrenamiento (training) y la muestra test (predicción). Esta técnica es muy frecuente tanto en problemas de estadística clásica como en Machine Learning e Inteligencia Artificial. Los datos de entrenamiento son utilizados para descubrir relaciones potencialmente predictivas y patrones escondidos en los datos, mientras que los datos de test se utilizan para evaluar la potencia y la utilidad de la predicción. Una vez obtenidos estos dos subconjuntos, el procedimiento a seguir es el siguiente: generamos un modelo predictivo con los datos de entrenamiento usando el algoritmo de clasificación que corresponda y aplicamos dicho modelo a los datos de test.

De esta manera podemos comparar los valores de la variable de baja real con las predicciones proporcionadas por el modelo. Este es el procedimiento que seguiremos en este proyecto para evaluar correctamente cada uno de los modelos creados. Para medir la habilidad haciendo predicciones de los clasificadores se hace uso de una tabla conocida como la matriz de confusión, que constituye la herramienta de visualización más empleada en aprendizaje supervisado. Su papel es comparar las predicciones del clasificador con los valores reales.

En nuestro caso, compararemos las clases que el modelo predice para cada cliente (baja o no baja) con el estado real de cada uno al final del trimestre. En la tabla 4.2 mostramos un ejemplo de la forma que adoptaría la matriz de confusión adaptado a nuestro problema.

	Real	
Predicción	NO	SI
NO	a	b
SI	c	d

Tabla 4.2. Ejemplo matriz de confusión

Las filas de la matriz de la tabla 4.2 representan el número de predicciones de cada clase (NO/SI), mientras que las columnas representan las instancias en la clase real (NO/SI). A partir de esta tabla es posible calcular diferentes medidas de calidad de las predicciones como la sensibilidad o la especificidad, que son medidas estadísticas obtenidas tras la realización de una prueba de clasificación binaria. Existe una gran diversidad de métricas además de estas y en la siguiente sección hablaremos más en profundidad de cada una de ellas.

4.2.1. Métricas de clasificación

Observar la matriz de confusión una vez hechas las predicciones puede parecer algo insuficiente a la hora de medir la eficiencia de un modelo clasificando datos. Una buena manera de explicar esto sería obtener una serie de medidas cuantitativas que nos permitan juzgar con argumentos sólidos el buen comportamiento de los clasificadores. A continuación detallamos algunas de las más importantes.

Accuracy

La accuracy mide la frecuencia con la que el clasificador hace una predicción correcta. Es la relación entre el número de predicciones correctas y el número total de predicciones. Su fórmula general es la siguiente:

$$\text{Accuracy} = \frac{\text{Predicciones correctas}}{\text{Individuos totales}}$$

y que respecto a la matriz mostrada en la tabla 4.2, se calcularía de la siguiente forma:

$$\text{Accuracy} = \frac{a+d}{a+b+c+d}.$$

A simple vista, la accuracy puede parecer una buena medida, sin embargo, no hace ninguna distinción entre las clases ya que las respuestas correctas para la clase NO y la clase SI son tratadas por igual. Esto se acentúa cuando el número de instancias que pertenecen a una clase es muy superior a otra. Por ejemplo, supongamos que estamos ante un problema con 2 clases desbalanceadas en las que contamos con un total de 9990 ejemplos de la clase 1 y 10 ejemplos de la clase 2. Si el modelo de clasificación siempre dice que los ejemplos son de la clase 1, tendríamos un accuracy del 99,9 % pero esto no ocurre porque el predictor sea muy preciso, sino porque no es capaz de detectar ejemplos de la clase 2.

Este es el motivo por el que la accuracy no es una buena medida de calidad en el caso de la predicción del churn, ya que el porcentaje de registros de baja (SI) es muy inferior al porcentaje de registros de no baja (NO).

Precisión y sensibilidad

La precisión y la sensibilidad son dos medidas diferentes, pero a menudo se utilizan conjuntamente. La precisión representa el porcentaje de casos positivos predichos correctamente, en nuestro caso, el porcentaje de bajas clasificadas de forma correcta. Por otro lado, la sensibilidad representa la capacidad para detectar una condición correctamente, en nuestro caso, corresponde a la capacidad para detectar una baja.

$$\text{Precision} = \frac{\text{Verdaderos positivos}}{\text{Nº de positivos predichos}}$$

$$\text{Sensibilidad} = \frac{\text{Verdaderos positivos}}{\text{Positivos reales}}$$

que aplicadas al ejemplo de la matriz de confusión mostrada en la tabla 4.1, se calcularía de la siguiente forma:

$$\text{Precision} = \frac{d}{c+d} \qquad y \qquad \text{Sensibilidad} = \frac{d}{a+d}.$$

Estas dos medidas son muy útiles respecto al problema que en este problema se plantea, ya que al operador de telecomunicaciones le interesa que nuestras predicciones de los clientes que se van a dar de baja sean lo más certeras posibles (alta precisión) así como que dicho número de predicciones sea razonable (sensibilidad). Por ejemplo, si predecimos 100 bajas y acertamos todas tendríamos una precisión del 100 % aunque este resultado sería inútil para el operador ya que la sensibilidad es muy pequeña puesto que el número de bajas real es de miles. Por lo tanto, la clave de este problema será encontrar un equilibrio entre ambas medidas.

Especificidad

La especificidad se refiere a la capacidad del modelo para excluir correctamente una condición, en nuestro caso, corresponde a la capacidad para detectar un cliente que no se da de baja. Matemáticamente, esto también puede ser escrito como:

$$\text{Especificidad} = \frac{\text{Verdaderos negativos predichos}}{\text{Negativos reales}}$$

Si nos basamos en la matriz de confusión mostrada en la tabla 4.2, la especificidad se calcularía como sigue:

$$\text{Especificidad} = \frac{a}{a+c}.$$

F-Score

Es la medida de precisión que tiene un test y suele emplearse en la fase de pruebas de algoritmos de búsqueda y recuperación de información y clasificación de documentos. El valor F [22] se considera como una media armónica que combina los valores de la precisión y de la sensibilidad y que puede calcularse de forma general de la siguiente forma:

$$\mathrm{F}_\beta = (1+\beta^2)\frac{\text{Precision} \cdot \text{Sensibilidad}}{(\beta^2 \cdot \text{Precision}) + \text{Sensibilidad}}.$$

En nuestro caso, la fórmula se reducirá a la siguiente expresión:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Sensibilidad}}{\text{Precision} + \text{Sensibilidad}}$$

en donde tomaremos β igual a uno, otorgándole la misma importancia o ponderación a la precisión y a la sensibilidad. Si β es mayor que uno, le daríamos mayor importancia a la sensibilidad, mientras que si es menor que uno se le da más importancia a la precisión. El rango de valores de esta métrica de clasificación oscila entre 0 (correspondería al peor caso) y 1 (correspondería al mejor caso).

La cuestión que nos planteamos es pensar en porqué una media armónica de dos métricas como la precisión y la sensibilidad nos puede resultar útil para decidir sobre la eficiencia de un clasificador. Por un lado, la media aritmética es mejor cuando tengamos cosas que tienen sentido sumar. Por otro lado, la media geométrica es mejor en el caso de la multiplicación de factores como pueden ser por ejemplo el cálculo de la tasa de interés en el tiempo.

En el caso de la media armónica, suele utilizarse para situaciones en las que estemos tratando con tasas o ratios. Como en nuestro problema, la precisión y la sensibilidad representan dos ratios distintos, la media armónica resulta la más apropiada. Su principal utilidad reside en búsqueda del equilibrio entre la precisión y la sensibilidad de un clasificador, es decir, en nuestro problema buscaremos un clasificador que haga un número razonable de predicciones y que éstas se ajusten lo máximo posible a la realidad de los datos.

AUC

Otra de las medidas más utilizadas en problemas de clasificación y que se caracteriza por ser algo más gráfica es calcular el valor bajo la curva ROC (Receiver Operating Characteristic), es decir, el AUC (Area Under Curve). La curva ROC [14] muestra la sensibilidad del clasificador por el trazado de la tasa de verdaderos positivos respecto a la tasa de falsos positivos. En otras palabras, muestra cómo se pueden obtener muchas clasificaciones positivas correctas cuando se permiten más y más falsos positivos. El clasificador perfecto, es decir, aquel que no comete errores, tendría una tasa de verdaderos positivos del 100 %, sin incurrir en ningún falso positivo aunque esto casi nunca sucede en la práctica.

La curva ROC es una curva que proporciona información detallada sobre el comportamiento del clasificador y a veces puede ser bastante resolutivo a nivel gráfico, por ejemplo, en la Figura 4.2.1 podemos observar tres tipos diferentes de curvas ROC y en las que se observa diferencias muy evidentes. Sin embargo, no siempre resulta una tarea fácil comparar dos curvas diferentes. En ocasiones resulta necesario tener una puntuación cuantificable en lugar de un gráfico que suele requerir de una inspección visual como criterio de decisión. El AUC es una manera de resumir la curva ROC en un solo número, de modo que se puedan comparar, fácilmente y de forma automática, dos curvas distintas.

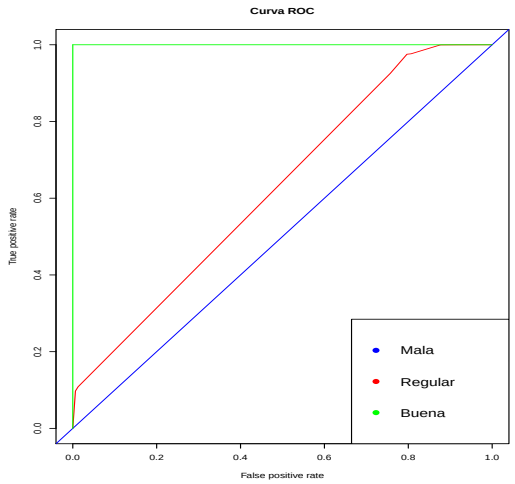


Figura 4.2.1. Ejemplos de la curva ROC

Una buena curva ROC guardará debajo de si misma un gran espacio (ya que la tasa de verdaderos positivos se dispara hasta el 100 % muy rápidamente) mientras que una curva ROC mala cubrirá una región más pequeña. Es decir, cuanto mayor sea el valor del AUC, mejor es la clasificación realizada.

La curva ROC se desarrolló por ingenieros eléctricos para medir la eficacia en la detección de objetos enemigos en campos de batalla mediante pantallas de radar, a partir de lo cual se desarrolla la Teoría de Detección de Señales (TDS). El análisis ROC se aplicó posteriormente en medicina, radiología, psicología y otras áreas durante varias décadas. Sólo recientemente ha encontrado aplicación en áreas como aprendizaje automático (o machine learning en inglés), y minería de datos (data mining en inglés).

4.2.2. Tablas de evaluación de los modelos

Hemos visto varias medidas que nos van a resultar muy útiles a la hora de comparar cada uno de los modelos que vamos a construir, aunque no solo nos vamos a basar en estas métricas como criterio de decisión para los clasificadores. En general, nos basaremos en las tablas de validación del modelo, el número de predicciones obtenidas por el mismo y las predicciones a futuro. Cada una de ellas se detalla a continuación, y se acompañarán de ejemplos específicos a fin de conseguir una mejor comprensión.

- **Tabla de validación:** Se corresponde con la matriz de confusión que nace de una validación cruzada entre las predicciones realizadas por el modelo y los valores reales de la baja objetivo. De esta forma, obtendremos una tabla como la que a continuación se muestra:

	Real		
Predicción	NO	SI	**Total**
NO	131734	3030	134764
SI	95	455	550
Total	131829	3485	135314

Tabla 1. Matriz de validación

  En este ejemplo concreto, contaríamos con una cantidad de 455 bajas bien clasificadas. El desbalanceo de clases se hace claramente visible en el número de predicciones obtenidas para cada clase.

- **Número de predicciones:** Esta tabla muestra el número de predicciones obtenidas para cada una de las dos clases, a saber, los clientes que sí se dan de baja y los que no. Se obtiene como resultado de aplicar el mismo modelo sobre los datos de la muestra test. De esta forma, obtendríamos una tabla como la que a continuación se muestra de ejemplo:

	NO	SI
Nº de predicciones	201345	2759

Tabla 2. Número de predicciones

  Podemos apreciar como el desbalanceo de clases deriva en la extremada diferencia sobre el número de predicciones obtenidas para cada clase.

- **Tabla a futuro:** En este caso, haremos una validación cruzada entre los valores reales de la baja objetivo del trimestre actual con las predicciones obtenidas por el modelo en el anterior trimestre. Un ejemplo de ello sería la tabla que sigue:

	Real			
Predicción	NO	SI	?	**Total**
NO	26859	937	434	28230
SI	169169	5201	1684	176054
?	114	6	2	122
Total	196142	6144	2120	204406

Tabla 3. Matriz a futuro

Hemos de notar que la última fila y/o columna representa los valores perdidos [44] durante la clasificación (también conocido como *missings*) y que pueden corresponderse, por ejemplo, con clientes que se dan de alta en el siguiente trimestre y que como no pertenecían a la muestra en primera instancia, no han podido ser clasificados en el futuro. En el caso real, esto puede hacer alusión a personas como los estudiantes que típicamente se afilian a contratos temporales en una determinada compañía durante su estancia académica fuera de su ciudad natal. Como podemos ver en este caso hay un total de 122 casos no clasificados por nuestro modelo, y esta cifra varía según la técnica empleada para corregir el desbalanceo de las clases.

Con esto cerramos el desarrollo en cuanto a la evaluación de nuestros modelos y en la siguiente sección presentaremos los resultados obtenidos por cada uno de ellos. Para una mejor comprensión elaboraremos una serie de tablas en donde recogeremos las distintas métricas de clasificación que hemos explicado en la sección 4.2.1 y que nos valdrán como criterio de decisión entre los distintos modelos que hemos ensayado.

4.3. Resultados

Una vez conocida la metodología y el funcionamiento de cada uno de los modelos, mostraremos los resultados obtenidos por los mismos con el fin de poder decantarnos por alguno de ellos. Con esto no nos referimos sólo al aspecto numérico, sino también a otras cuestiones que han ido surgiendo durante el análisis de las técnicas de desbalanceo abordadas en este trabajo y que a continuación explicamos.

4.3.1. J48 vs rpart

Existen varias funciones en el lenguaje de *R* que nos permiten construir árboles de decisión. Una de las más empleadas es la función *rpart* (*recursive partitioning*) que construye un árbol de decisión centrando su atención en los valores perdidos. Sin embargo, en la práctica, hemos decidido utilizar la función *J*48 de la librería *RWeka* como clasificador estadístico por varias razones:

- Esta función es la que tiene implementada KNIME (herramienta utilizada por la consultora para la construcción de árboles de decisión).
- El *rpart* es más costoso a la hora de obtener las reglas que van asignando las probabilidades de baja a cada cliente.
- La función *rpart* convierte a binario las variables factor y esto supone un problema a la hora de obtener el número de bajas predichas por el modelo.
- Los resultados obtenidos por la función *J*48 superaban, en general, a los obtenidos por *rpart*.

Método	**Precisión**	**Sensibilidad**	**F-Score**	**AUC**	**Nº de predicciones**
Datos brutos (rpart)	0.8918	0.0617	0.1154	0.6469	-
Datos brutos (J48)	0.9223	0.1163	0.2065	0.6356	52
$1-50$ (rpart)	0.6820	0.0763	0.1372	0.6661	-
$1-50$ (*J*48)	0.7054	0.1532	0.2517	0.7115	468
$2-100$ (rpart)	0.6844	0.0771	0.1385	0.6661	-
$2-100$ (*J*48)	0.6994	0.1410	0.2347	0.6993	618
$3-100$ (rpart)	0.3837	0.1515	0.2172	0.6810	-
$3-100$ (*J*48)	0.2907	0.1871	0.2277	0.6450	1584
$2-50$ (rpart)	0.3826	0.1520	0.2175	0.6810	-
$2-50$ (*J*48)	0.4930	0.1834	0.2552	0.6891	4052
$3-80$ (rpart)	0.6844	0.0771	0.1385	0.6661	-
$3-80$ (*J*48)	0.1975	0.1917	0.1945	0.6335	1912

Tabla 4.3.1. Comparación de las funciones J48 y rpart.

Hemos de mencionar que la notación empleada en la Tabla 4.3.1 corresponde a la expresión del número de réplicas de la clase minoritaria (primera cifra) y el porcentaje de casos seleccionados de la clase mayoritaria (segunda cifra). De esta forma, 1-50, por ejemplo, indicará una selección bruta de los casos de la clase minoritaria al que se añade un 50 % de los casos de la mayoritaria.

4.3.2. Algoritmos de resampling

En la siguiente tabla se recogen los resultados obtenidos para las distintas pruebas realizadas en los algoritmos de resampling. En la primera fila contamos con las soluciones que proporciona el modelo clasificatorio sin modificar la distribución de los datos (datos brutos) con el fin de compararlos con los demás modelos en donde se balancea cada una de las clases (tanto de forma simultánea como alterna). Para los casos de undersampling se especifica el porcentaje de casos seleccionados sobre la muestra de la clase mayoritaria, mientras que en el oversampling se mostrará el número de réplicas de la clase minoritaria efectuadas antes de aplicar el modelo. Para la construcción de los algoritmos híbridos se ha empleado la misma notación que en la sección 4.3.1 que acabamos de ver.

Método		**Precisión**	**Sensibilidad**	**F-Score**	**AUC**	**Nº Predicciones**
Datos brutos		0.9223	0.1163	0.2065	0.6356	52
Undersampling	15 %	0.3152	0.2241	0.2619	0.7131	5313
	25 %	0.5027	0.1847	0.2701	0.7123	4035
	50 %	0.8574	0.1328	0.2299	0.6281	67
	75 %	0.8440	0.1242	0.2165	0.6271	58
	90 %	0.8617	0.1234	0.2159	0.6274	51
Oversampling	x2	0.6994	0.1410	0.2347	0.6993	618
	x3	0.2907	0.1871	0.2277	0.6450	1584
	x4	0.1523	0.2074	0.1756	0.5541	2838
	x5	0.1227	0.2106	0.1550	0.5643	1961
	x8	0.1167	0.2321	0.1553	0.5904	5484
Algoritmos híbridos	$2-50$	0.4930	0.1834	0.2552	0.6891	4052
	$3-80$	0.1975	0.1917	0.1945	0.6335	1912
	$4-25$	0.0690	0.3836	0.1169	0.6198	32400
	$5-15$	0.0689	0.4258	0.1186	0.6323	41194

Tabla 4.3.2. Resultados de los algoritmos de resampling.

Tras los resultados mostrados en la tabla 4.3.3, podemos comentar varios aspectos sobre los algoritmos de resampling:

- La máxima precisión obtenida ocurre cuando no se modifica la distribución de los datos, aunque existen casos en los que se han conseguido clasificadores muy precisos balanceando las clases, como por ejemplo, utilizando la técnica de undersampling, que permite equilibrar la cantidad de registros de cada clase al reducir en más de un 50 % los casos de la clase mayoritaria.
- En general, la sensibilidad de los clasificadores es baja ya que al haber pocos casos de la clase minoritaria en la muestra, la capacidad para detectarlos por el clasificador será también escasa.
- Los valores del AUC oscilan entre 0,55 y 0,75, y en las técnicas de undersampling vuelve a encontrar sus mejores cifras, lo cual indica que en estos casos, el clasificador suele hacer una mejor clasificación de los datos.
- El porcentaje de baja presente en los datos se sitúa en torno al 3 %. Esta cifra representa el valor al que debe aproximarse el número de predicciones de nuestro clasificador, y en base a esto, podemos observar que los algoritmos híbridos $3-80$ y la técnica de oversampling (quintuplicando los casos de la clase minoritaria) son los más apropiados.

Tras las recientes investigaciones y los resultados obtenidos en este ensayo, se han recomendado ciertos límites para proceder con el desbalanceo de los datos. En general, se propone no modificar la distribución de los datos de forma que se supere el 15 % de baja presente en los mismos. Esto puede derivar en métricas desfavorables, como por ejemplo, el número de predicciones obtenidas es excesivamente elevado o se obtiene un clasificador poco preciso.

4.3.3. Algoritmos multiclasificadores

En esta sección discutiremos los resultados proporcionados por los algoritmos multiclasificadores que se han visto hasta ahora en función de los distintos valores que hemos asignado a cada uno de sus parámetros.

Boosting

Nº Iteraciones	Precisión	Sensibilidad	F-Score	AUC	Nº Predicciones
2	0.9503	0.0858	0.1573	0.5613	1037
3	0.9650	0.0962	0.1749	0.6146	24
5	0.9905	0.1047	0.1893	0.7076	1026
8	0.9057	0.0862	0.1574	0.7409	1003
10	0.8446	0.0867	0.1572	0.7513	32

Tabla 4.3.2.1. Resultados del algoritmo boosting.

Como cabía esperar, este método constituye un algoritmo de clasificación muy preciso con un porcentaje superior al 85 % en cuanto a bajas predichas correctamente.

Sin embargo, el número de predicciones obtenido no se aproxima demasiado al valor real (dos mil bajas predichas), además de contar con una escasa capacidad para detectar la presencia de una baja ya que no ataca de forma directa el problema del desbalanceo presente en nuestra muestra de datos.

Por otro lado, a medida que aumentamos el número de iteraciones, desciende la precisión del clasificador ya que el propio algoritmo se va centrando en aquellos casos más difíciles de clasificar a medida que va reponderando los pesos en cada iteración. A pesar de ello, el porcentaje de datos bien clasificados aumenta a medida que introducimos más iteraciones, y este hecho se refleja en el incremento paulatino del valor AUC mostrado en la quinta columna de la tabla 4.3.2.1, donde pasa de un valor inicial de 0,5613 a un valor final de 0,7513.

Bagging

A continuación mostramos los resultados obtenidos por el método bagging en función del número de árboles construidos en cada caso. Notar que se han realizado las pruebas para la construcción de un número de árboles superior al mostrado en la Tabla 4.3.2.2, sin embargo, el algoritmo iba perdiendo practicidad a medida que incrementábamos este valor, con lo cual sólo expondremos los resultados obtenidos hasta un número máximo de 20 árboles.

Nº árboles	**Precisión**	**Sensibilidad**	**F-Score**	**AUC**	**Nº Predicciones**
2	0.9609	0.0777	0.1617	0.5416	86
5	0.8417	0.0789	0.1456	0.5531	83
10	0.9686	0.0708	0.1319	0.5544	1014
20	0.9641	0.0771	0.1427	0.5522	1012

Tabla 4.3.2.2. Resultados del algoritmo bagging.

Podemos observar que el método bagging no se ve muy afectado por el número de subclasificadores construidos para elaborar el clasificador final. Como cabía esperar, también se obtiene un algoritmo muy preciso con tasas superiores al 80 % de bajas correctamente predichas y con una escasa sensibilidad ya que no se corrige el problema del desbalanceo de clases presente en la muestra de datos. Los valores AUC son bastante parejos (alrededor de 0,55) atendiendo al distinto número de subclasificadores construidos. Sin embargo, un rasgo distintivo entre todos estos modelos que se han ensayado, radica en el número de predicciones obtenido por los mismos en donde la balanza se decanta por los dos últimos (utilizan 10 y 20 subclasificadores respectivamente para elaborar el clasificador final) que genera un total de bajas predichas que se acercan al valor esperado.

Random Forest

Para llevar este método a la práctica es necesario tener en cuenta varios detalles. Como hemos visto en la sección 3.1.2, hay varios parámetros que intervienen en la construcción de un modelo clasificador a través del método Random Forest. A saber, tanto el número de árboles construidos como el número de variables empleadas pueden jugar un papel importante en la eficiencia del clasificador. Por ello, en la siguiente tabla recogeremos los distintos resultados obtenidos en función de los diversos valores asignados a estos dos parámetros y, según esto, haremos las conclusiones pertinentes una vez comparado cada uno de los modelos que se han ensayado.

Nº variables	Nº árboles	Precisión	Sensibilidad	F-Score	AUC	Nº predicciones
2	2	0.1045	0.0353	0.0527	0.5280	7594
	5	0.3333	0.0017	0.0034	0.5421	499
	10	0.2500	0.0003	0.0006	0.5816	4
	50	-	-	-	-	-
	100	-	-	-	-	-
8	2	0.1433	0.1489	0.1460	0.5877	11066
	5	0.5104	0.1059	0.1754	0.6274	2624
	10	0.8061	0.0989	0.1761	0.6571	1072
	50	0.9335	0.0846	0.1551	0.7066	92
	100	0.9495	0.0863	0.1582	0.7155	141
30	2	0.1191	0.1658	0.1386	0.5965	14535
	5	0.5056	0.1426	0.2224	0.6361	14964
	10	0.5977	0.1228	0.2037	0.6541	5360
	50	0.7965	0.1179	0.2054	0.7133	3536
	100	0.8152	0.1165	0.2039	0.7143	2981
60	2	0.1018	0.1653	0.1260	0.5896	19785
	5	0.4358	0.1354	0.2066	0.6225	10429
	10	0.5573	0.1380	0.2212	0.6579	12286
	50	0.7475	0.1299	0.2213	0.7047	5069
	100	0.7671	0.1285	0.2201	0.7154	6154

Tabla 4.3.2.3. Resultados del algoritmo RandomForest.

En primer lugar, comentaremos que en el primer caso (dos variables utilizadas para la construcción del modelo), no se han obtenido predicciones para la clase minoritaria si utilizamos más de diez árboles, ya que entendemos que este número de variables es insuficiente para hacer un promedio del número de predicciones obtenidas por cada subclasificador.

Se pueden apreciar varios detalles observando los resultados recogidos en la tabla 4.3.2.3:

- La precisión del clasificador aumenta a medida que incrementamos el número de subclasificadores construidos.
- Los clasificadores más precisos se alcanzan en aquellos modelos que utilizan un total de 8 variables, ya que se llegan a alcanzar tasas de precisión por encima del 90 %, algo que no se logra en ningún otro caso más.
- Dado que el número de bajas presentes en la muestra es muy escaso, la sensibilidad de los métodos será muy baja.
- El número de predicciones obtenidas desciende a medida que incrementamos el número de árboles construidos para la confección del clasificador final.

AdaOUBoost

Este multiclasificador es el único de los que hemos visto que modifica la distribución de los datos, y que por tanto, ataca el problema del desbalanceo antes de proceder con la clasificación. A continuación recogemos los resultados obtenidos por el mismo en función de los distintos clusters en los que se ha realizado la división de la clase mayoritaria además de la técnica smote aplicada en cada caso.

Nº clusters	Precisión	Sensibilidad	F-Score	AUC	Nº de predicciones
2	0.0289	0.8932	0.0590	0.6945	88962
3	0.0276	0.8252	0.0534	0.6857	121541
5	0.0261	0.9257	0.0570	0.6513	144745
8	0.0258	0.9974	0.0503	0.6512	157668
10	0.0257	1	0.0503	0.5981	162418

Tabla 4.3.2.4. Resultados para el algoritmo AdaOUBoost.

Notar que para este caso, hemos aplicado la técnica smote haciendo un incremento del dos mil por ciento de la clase minoritaria y una reducción del doscientos por ciento de la clase mayoritaria por cada uno de los casos de la minoritaria generados. Esto supone un balanceo muy favorable en torno a la clase minoritaria, lo que justifica el elevado número de predicciones obtenida por el modelo, como se puede apreciar en la última columna de la tabla 4.3.2.4. En base a esta forma de remuestrear los datos, la precisión del modelo es muy baja ya que sigue habiendo un elevado número de casos de la clase mayoritaria lo que justifica también los elevados valores de sensibilidad obtenidos.

4.3.4. Comparación final

En base a todos los resultados obtenidos hasta ahora, ya tenemos una perspectiva general del comportamiento de nuestro modelos de clasificación atendiendo tanto a su funcionamiento como a la presencia del desbalanceo de las clases. Para poder establecer una cómoda comparativa entre todos ellos recogeremos en la Tabla 4.3.4 gran parte de los resultados conseguidos por estos algoritmos. Las métricas de clasificación empleadas para esta tarea son las mismas que se han utilizado hasta ahora para el resto de modelos: precisión, sensibilidad, valor F, valor AUC y número de predicciones obtenidas para la clase minoritaria.

Método	**Precisión**	**Sensibilidad**	**F-Score**	**AUC**	**Nº de predicciones**
Datos brutos	0.8606	0.1222	0.2140	0.6256	52
$1-50$	0.7054	0.1532	0.2517	0.7115	468
$2-100$	0.6994	0.1410	0.2347	0.6993	618
$3-100$	0.2907	0.1871	0.2277	0.6450	1584
$3-80$	0.1975	0.1917	0.1945	0.6335	1912
$2-50$	0.4193	0.1834	0.2552	0.6891	4052
AOUB($k=2$)	0.0289	0.8932	0.0559	0.6945	88962
AOUB($k=3$)	0.0276	0.8252	0.0534	0.6857	121541
AOUB($k=5$)	0.0261	0.9257	0.0570	0.6513	144745
AOUB($k=8$)	0.0258	0.9974	0.0503	0.6512	157668
Bagging (2 árb.)	0.9609	0.0777	0.1617	0.5416	86
Bagging (10 árb.)	0.9686	0.0708	0.1319	0.5544	1014
Bagging (20 árb.)	0.9641	0.0771	0.1427	0.5522	1012
Boosting (2 árb.)	0.9503	0.0858	0.1574	0.5613	1037
Boosting (5 árb.)	0.9905	0.1047	0.1893	0.7076	1026
Boosting (10 árb.)	0.8446	0.0867	0.1572	0.7513	32
RF (2 árb. + 8 var.)	0.1433	0.1489	0.1460	0.5877	11066
RF (5 árb. + 8 var.)	0.5104	0.1059	0.1754	0.6274	2624
RF (10 árb. + 8 var.)	0.8061	0.0989	0.1761	0.6571	1072
RF (50 árb. + 8 var.)	0.9335	0.0846	0.1551	0.7066	92
RF (100 árb. + 8 var.)	0.9495	0.0863	0.1582	0.7155	141

Tabla 4.3.4. Comparación de los resultados de todos los métodos descritos.

Lo que se pretende es encontrar aquel método con una serie de métricas favorables, es decir, un buena precisión en la clasificación, un elevado porcentaje de clasificación correcta (AUC alto) y un número de predicciones que se aproxime al valor real presente en la muestra (en este caso, en torno a las 2000 predicciones) que oscila sobre el 3 % de baja.

En la tabla se marcan de color rojo aquellas técnicas que proporcionan los mayores valores en cada una de las métricas utilizadas para evaluar cada uno de los modelos. Podemos apreciar varias cuestiones:

- Los algoritmos boosting y Random Forest constituyen los métodos más precisos en la clasificación.
- Los métodos AdaOUBoost y bagging presentan la mejor capacidad para detectar una baja correctamente.
- Los algoritmos que remuestrean los datos (resampling) son los que poseen el mayor F-Score.
- En general, los valores para el AUC son bastante parejos aunque algunos métodos como el resampling $1-50$ o el método boosting (construyendo 10 árboles) se desmarcan un poco más que el resto.
- El número de predicciones obtenido en el método de AdaOUBoost es claramente elevado, y ocurre como consecuencia de la modificación realizada en la distribución de los datos. Sin embargo, aunque es el algoritmo que más bajas predice, no es el más apropiado ya que en la realidad no se da esa situación. Otros métodos como el resampling $3-80$, hace una buena aproximación del número de bajas predichas.

4.3.5. Modelos propuestos

Hemos visto muchas técnicas que, de forma individual, pueden funcionar bien ante un problema habitual como el desbalanceo de clases. Pero podemos pararnos a pensar en qué ocurriría si, combinando estas técnicas, somos capaces de mejorar los resultados. Esta idea será la que vamos a desarrollar en esta sección, en particular combinaremos las técnicas de resampling vistas hasta ahora con alguno de los multiclasificadores más destacados. En la tabla 4.3.5.1 recogemos los mejores resultados proporcionados por cada uno de los métodos que hemos visto hasta ahora y que nos ayudarán a justificar la elección de los modelos híbridos que en esta sección se describen:

Método	**Precisión**	**Sensibilidad**	**F-Score**	**AUC**	**Nº de predicciones**
$1-50$	0.7054	0.1532	0.2517	0.7115	468
$3-80$	0.1975	0.1917	0.1945	0.6335	1912
$2-50$	0.4193	0.1834	0.2552	0.6891	4052
AOUB($k=8$)	0.0258	0.9974	0.0503	0.6512	157668
Bagging (10 árb.)	0.9686	0.0708	0.1319	0.5544	1014
Boosting (5 árb.)	0.9905	0.1047	0.1893	0.7076	1026
RF (10 árb. + 8 var.)	0.8061	0.0989	0.1761	0.6571	1072

Tabla 4.3.5.1. Resumen de los mejores resultados obtenidos.

Como podemos observar, hemos seleccionado las mejores técnicas en función de las distintas métricas utilizadas para su evaluación. Veamos qué ocurre si combinarlos puede derivar en una mejora del rendimiento del modelo en cuanto a la clasificación. Para ello, expondremos en primer lugar cuáles son lo modelos híbridos con los que ensayaremos y, en segundo lugar, qué resultados obtendremos a partir de ellos.

1. **Resampling + Random Forest**: Utilizaremos técnicas de undersampling y oversampling para balancear la distribución de los datos y luego, el multiclasificador Random Forest para obtener el clasificador final. En concreto:
   - Triplicaremos el número de instancias de la clase minoritaria y reduciremos al 80 % el número de casos de la clase mayoritaria. También haremos el resampling $2-50$, es decir, duplicaremos los casos de la clase minoritaria y reduciremos al 50 % los casos de la mayoritaria.
   - Aplicaremos el algoritmo Random Forest construyendo 10 subclasificadores y empleando 8 variables en cada caso.
2. **Resampling + Boosting**: Utilizaremos las técnicas de undersampling y oversampling para balancear la distribución de los datos y posteriormente, emplearemos el método boosting para obtener el clasificador final. En concreto:
   - Triplicaremos el número de instancias de la clase minoritaria y reduciremos al 80 % el número de casos de la clase mayoritaria. También haremos resampling $1-50$, es decir, haremos una selección bruta de los casos de la clase minoritaria y los combinaremos con una reducción del 50 % de la mayoritaria.
   - Aplicaremos el algoritmo boosting construyendo dos subclasificadores, con los que obtendremos el clasificador final.
3. **AdaOUBoost + Random Forest**: Emplearemos las técnicas de remuestreo que utiliza el algoritmo AdaOUBoost que hemos revisado en este trabajo para construir después el clasificador final mediante el algoritmo multiclasificador Random Forest. En concreto:
   - Utilizaremos la técnica SMOTE para remuestrear los datos. En particular, incrementaremos en un 200 % los casos de la clase minoritaria y, por cada uno de estos ejemplos generados, reduciremos 200 instancias de la clase mayoritaria.
   - Aplicamos el algoritmo Random Forest promediando 10 subclasificadores y utilizando 8 variables para cada caso.

Método	**Precisión**	**Sensibilidad**	**F-Score**	**AUC**	**Nº de predicciones**
3-80 + RF	0.3814	0.1213	0.1840	0.6400	5301
3-80 + Boosting	0.7543	0.1521	0.2531	0.7464	2924
AOUB + RF	0.0375	0.7216	0.0713	0.6807	107539
2-50 + RF	0.2682	0.1526	0.1945	0.6534	8321
1-50 + Boosting	0.8160	0.1017	0.1808	0.7194	686

Tabla 4.3.5.2. Resultados para los modelos híbridos propuestos.

Podemos observar que, en general se mejoran ciertas métricas de clasificación respecto a las obtenidas de formas individual, sin embargo, en un aspecto más global no parece haber una mejora significativa. Por ejemplo, comparemos el algoritmo $3-80$ con el modelo híbrido que combina precisamente esta técnica con el método boosting.

Método	**Precisión**	**Sensibilidad**	**F-Score**	**AUC**	**Nº de predicciones**
3-80 + RF	0.3814	0.1213	0.1840	0.6400	5301
3-80	0.1975	0.1917	0.1945	0.6335	1912

Tabla 4.3.5.3: Comparación del algoritmo resampling y su combinación con el Random Forest.

Podemos apreciar que el modelo híbrido consigue mejorar la precisión obtenida por el algoritmo de resampling particular, pero se aleja del número de predicciones al que debe aproximarse según la muestra de datos. En este caso, el algoritmo $3-80$ desempeña un mejor papel.

Veamos ahora otro caso, en concreto, compararemos el modelo híbrido correspondiente al resampling $3-80$ más el algoritmo boosting, con sus algoritmos particulares.

Método	**Precisión**	**Sensibilidad**	**F-Score**	**AUC**	**Nº de predicciones**
3-80 + Boosting	0.7543	0.1521	0.2531	0.7464	2924
3-80	0.1975	0.1917	0.1945	0.6335	1912
Boosting	0.9503	0.0858	0.1573	0.5613	1037

Tabla 4.3.5.4: Comparación del boosting y algoritmo de resampling con la hibridación de ambos.

Podemos observar que el modelo híbrido no es tan preciso como el método boosting, sin embargo el porcentaje de clasificación correcta es superior si combinamos ambos modelos (0,7464) que utilizando los algoritmos particulares (0,6335 y 0,5613 respectivamente). En cuanto al número de bajas predichas, el algoritmo $3-80$ se aproxima más al caso real ya que estima alrededor de 2000, mientras que el modelo híbrido excede en casi 1000 unidades este valor (2924).

El resto de modelos presentan carencias similares a las que hemos reflejado ahora, con lo cual resulta difícil decantarse por uno de ellos. En la última sección mostraremos las conclusiones en base a todo lo ocurrido tras estas pruebas.

Capítulo 5

Conclusiones

Después de haber efectuado todas las pruebas pertinentes sobre los modelos de clasificación y las técnicas de desbalanceo desarrolladas en este trabajo, podemos elaborar una serie de conclusiones desde dos puntos de vista diferentes: uno desde el aspecto estadístico y otro desde un enfoque más empresarial. Atendiendo a ambas, se obtienen varias deducciones que a continuación enumeramos:

- Se han obtenido unas métricas favorables en el algoritmo Random Forest, sin embargo, su difícil interpretabilidad lo descarta como modelo clasificatorio ante el problema del desbalanceo, ya que a la empresa le interesa obtener las reglas por las cuales se asignan las probabilidades de baja a cada cliente.
- Los modelos híbridos propuestos en este trabajo no mejoran de forma notoria los resultados obtenidos por los modelos particulares que los constituyen.
- A nivel práctico, el algoritmo $3 - 80$ es un algoritmo fácil de implementar y que produce unos resultados bastante favorables en cuanto a la clasificación. Además considera el problema del desbalanceo de clases presente en la muestra y nos permite obtener las reglas que asigna a cada cliente para determinar si se da o no de baja.
- Se ha estudiado el distinto comportamiento que adopta el clasificador si modificamos los parámetros de su construcción. En el caso de los multiclasificadores, podemos ver que el número óptimo de variables para construir el modelo son un total de ocho, y el número de subclasificadores utilizados para generar el clasificador final son diez. Bajo estas condiciones, se aprecia una mejora en la precisión del modelo, como cabía esperar, en detrimento de otras métricas como el AUC o el número de predicciones obtenidas.

Existen diversos métodos y algoritmos que en este trabajo no se han revisado pero que pueden resultar de alto interés para la cuestión del desbalanceo de clases, por ejemplo, el modelo logístico o la profundización sobre los algoritmos de resampling entre otros. Algunos de ellos constaban de desarrollos metodológicos más complejos y con un tiempo de procesado superior a los que hemos visto en este trabajo, y por este tipo de razones, no han sido considerados en la elaboración de este proyecto.

Apéndice A

Código en R

```
library(RWeka)

## 1.DATOS BRUTOS ##

dat_NP=subset(training,training$Cuentas_Acc..Llamadas=="NO PERMITE")
dat_P=subset(training,training$Cuentas_Acc..Llamadas=="PERMITE")

# Modelo
formula=paste("Baja.objetivo", "~",paste(colnames(dat_NP)[2:71],collapse="+"))
Arbol<-J48(eval(parse(text=formula)),data=dat_NP)

# Predicción
Prediccion <- predict(Arbol,dat_P, type="class")
tabla=table(Prediccion,dat_P$Baja.objetivo);tabla

# Matriz de confusión
confusionMatrix(tabla,positive ="SI")

# Nº de predicciones
Prediccion2 <- predict(Arbol,prediccion,type="class")
table(Prediccion2)

# Matriz a futuro:
training=read.csv2("training_fijo_junio.csv",header = TRUE)
tab1=data.frame(prediccion$Producto.base_ID.Producto,Prediccion2)
tab2=data.frame(training$Producto.base_ID.Producto,training$Baja.objetivo)
library("dplyr")
colnames(tab1)=c("ID","Prediccion2")
colnames(tab2)=c("ID","Baja real")
join=full_join(tab1,tab2,by="ID")
tabla=table(join[,2],join[,3],useNA = "always");tabla
```

```
# Curva ROC y valor AUC:
Pred_roc <- predict(Arbol,dat_P, type="prob")
pred <- prediction(Pred_roc[,1],dat_P$Baja.objetivo,label.ordering = c("SI","NO"))
perf <- performance(pred,"tpr","fpr")
plot(perf,col="red",main="Curva ROC")
abline(a=0,b=1,main="Curva ROC")
performance(pred,"auc")

## 2.DUPLICACIÓN DE LA CLASE MINORITARIA ##

# Divido PERMITE/NO PERMITE:
formula=paste(colnames(training)[72], "~",paste(colnames(training)[2:71],collapse="+"))
dat_NP=subset(training,training$Cuentas_Acc..Llamadas=="NO PERMITE")
dat_P=subset(training,training$Cuentas_Acc..Llamadas=="PERMITE")

# Divido SI/NO:
dat_NP_SI=subset(dat_NP,dat_NP$Baja.objetivo=="SI")
dat_NP_NO=subset(dat_NP,dat_NP$Baja.objetivo=="NO")

# Duplico los SI:
datos_rep1=rbind(dat_NP_SI,dat_NP_SI)      #replicamos
datos_rep=rbind(datos_rep1,dat_NP_NO)      #unimos la replica con los NO

datos_rep=sample(datos_rep) #mezclamos los SI y los NO

# Modelo
Arbol<-J48(eval(parse(text=formula)),data=datos_rep)

# Predicciones
Prediccion <- predict(Arbol,dat_P, type="class")

# Matriz confusión (validación)
tabla=table(Prediccion,dat_P$Baja.objetivo);tabla

# Matriz de confusión
confusionMatrix(tabla,positive ="SI")

# N° de predicciones
Prediccion2 <- predict(Arbol,prediccion,type="class")
table(Prediccion2)

# Matriz a futuro:
training=read.csv2("training_fijo_junio.csv",header = TRUE)

tab1=data.frame(prediccion$Producto.base_ID.Producto,Prediccion2)
tab2=data.frame(training$Producto.base_ID.Producto,training$Baja.objetivo) #training junio
library("dplyr")
colnames(tab1)=c("ID","Prediccion2")
colnames(tab2)=c("ID","Baja real")
join=full_join(tab1,tab2,by="ID")
tabla=table(join[,2],join[,3],useNA = "always");tabla
```

```
# Curva ROC y valor AUC:
Pred_roc <- predict(Arbol,dat_P, type="prob")
pred <- prediction(Pred_roc[,1],dat_P$Baja.objetivo,label.ordering = c("SI","NO"))
perf <- performance(pred,"tpr","fpr")
plot(perf,col="red",main="Curva ROC")
abline(a=0,b=1,main="Curva ROC")
performance(pred,"auc")

## 3.REDUCCIÓN DE LA CLASE MAYORITARIA ##

# Dividimos PERMITE/NO PERMITE:
formula=paste(colnames(training)[72], "~",paste(colnames(training)[2:71],collapse="+"))
dat_NP=subset(training,training$Cuentas_Acc..Llamadas=="NO PERMITE")
dat_P=subset(training,training$Cuentas_Acc..Llamadas=="PERMITE")

# Dividimos SI/NO:
dat_NP_SI=subset(dat_NP,dat_NP$Baja.objetivo=="SI")
dat_NP_NO=subset(dat_NP,dat_NP$Baja.objetivo=="NO")

# Reducimos los NO:

# Seleccionamos el 50% de los NO:
n=length(dat_NP_NO[,1]);n
i=1:n
m=floor(n/2)
ii=sample(i,size=m,replace=FALSE)      #reduccion(sorteo)
datos_rep1=dat_NP_NO[ii,]
datos_rep=rbind(datos_rep1,dat_NP_SI)

datos_rep=sample(datos_rep)            #mezclamos los SI y los NO

# Modelo
Arbol<-J48(eval(parse(text=formula)),data=datos_rep)

# Predicciones
Prediccion <- predict(Arbol,dat_P, type="class")

# Matriz de confusion(validacion)
tabla=table(Prediccion,dat_P$Baja.objetivo);tabla

# Matriz de confusión
confusionMatrix(tabla,positive ="SI")

# N° de predicciones
Prediccion2 <- predict(Arbol,prediccion,type="class")
table(Prediccion2)

# Matriz a futuro:
training=read.csv2("training_fijo_junio.csv",header = TRUE)
```

```
tab1=data.frame(prediccion$Producto.base_ID.Producto,Prediccion2)
tab2=data.frame(training$Producto.base_ID.Producto,training$Baja.objetivo)
library("dplyr")
colnames(tab1)=c("ID","Prediccion2")
colnames(tab2)=c("ID","Baja real")
join=full_join(tab1,tab2,by="ID")
tabla=table(join[,2],join[,3],useNA = "always");tabla

# Curva ROC y valor AUC:
Pred_roc <- predict(Arbol,dat_P, type="prob")
pred <- prediction(Pred_roc[,1],dat_P$Baja.objetivo,label.ordering = c("SI","NO"))
perf <- performance(pred,"tpr","fpr")
plot(perf,col="red",main="Curva ROC")
abline(a=0,b=1,main="Curva ROC")
performance(pred,"auc")

## 4. COMBINACIÓN REDUCCIÓN CLASE MAY. + AUMENTO CLASE MIN.  ##

# Duplicamos la minoritaria y seleccionamos 30000 registros de la mayoritaria.

# Dividimos PERMITE/NO PERMITE:
formula=paste("Baja.objetivo", "~",paste(colnames(training)[2:71],collapse="+"))
dat_NP=subset(training,training$Cuentas_Acc..Llamadas=="NO PERMITE")
dat_P=subset(training,training$Cuentas_Acc..Llamadas=="PERMITE")

# Dividimos SI/NO:
dat_NP_SI=subset(dat_NP,dat_NP$Baja.objetivo=="SI")
dat_NP_NO=subset(dat_NP,dat_NP$Baja.objetivo=="NO")

# Aumentamos los SI:
datos_rep1=rbind(dat_NP_SI,dat_NP_SI)

# Reducimos los NO:

n=length(dat_NP_NO[,1]);n
i=1:n
ii=sample(i,size=30000,replace=FALSE)      #reduccion(sorteo)
datos_rep2=dat_NP_NO[ii,]
datos_rep=rbind(datos_rep1,datos_rep2)     #unimos los SI con los NO

datos_rep=sample(datos_rep)                #mezclamos los SI y los NO

# Modelo
Arbol<-J48(eval(parse(text=formula)),data=datos_rep)

# Prediccción en Test
Prediccion <- predict(Arbol,dat_P, type="class")

# Matriz de confusion(validacion)
tabla=table(Prediccion,dat_P$Baja.objetivo);tabla
```

```
# Matriz de confusión
confusionMatrix(tabla,positive ="SI")

# N° de predicciones
Prediccion2 <- predict(Arbol,prediccion,type="class")
table(Prediccion2)

# Matriz a futuro:
training=read.csv2("training_fijo_junio.csv",header = TRUE)

tab1=data.frame(prediccion$Producto.base_ID.Producto,Prediccion2)
tab2=data.frame(training$Producto.base_ID.Producto,training$Baja.objetivo) #training junio

library("dplyr")
colnames(tab1)=c("ID","Prediccion2")
colnames(tab2)=c("ID","Baja real")
join=full_join(tab1,tab2,by="ID")
tabla=table(join[,2],join[,3],useNA = "always");tabla

# Curva ROC y valor AUC:
Pred_roc <- predict(Arbol,dat_P, type="prob")
pred <- prediction(Pred_roc[,1],dat_P$Baja.objetivo,label.ordering = c("SI","NO"))
perf <- performance(pred,"tpr","fpr")
plot(perf,col="red",main="Curva ROC")
abline(a=0,b=1,main="Curva ROC")
performance(pred,"auc")

## OTROS CASOS DE COMBINACIONES DE AMBAS TÉCNICAS:3-80 Y 2-50 ##

# Triplicamos los SI:
datos_rep1=rbind(dat_NP_SI,dat_NP_SI)
datos_rep1=rbind(datos_rep1,dat_NP_SI)

# Reducimos al 80% los NO:

n1=52097 #80% de los NO
n=length(dat_NP_NO[,1]);n
i=1:n
ii=sample(i,size=n1,replace=FALSE)         #reduccion(sorteo)
datos_rep2=dat_NP_NO[ii,]
datos_rep=rbind(datos_rep1,datos_rep2)     #unimos los SI con los NO

datos_rep=sample(datos_rep)                #mezclamos los SI y los NO

# Modelo
Arbol<-J48(eval(parse(text=formula)),data=datos_rep)

# Prediccción en Test
Prediccion <- predict(Arbol,dat_P, type="class")

# Matriz confusión (validación)
tabla=table(Prediccion,dat_P$Baja.objetivo);tabla
```

```
# Matriz de confusión
confusionMatrix(tabla,positive ="SI")

# N° de predicciones
Prediccion2 <- predict(Arbol,prediccion,type="class")
table(Prediccion2)

# Matriz a futuro:
training=read.csv2("training_fijo_junio.csv",header = TRUE)

tab1=data.frame(prediccion$Producto.base_ID.Producto,Prediccion2)
tab2=data.frame(training$Producto.base_ID.Producto,training$Baja.objetivo) #training junio

library("dplyr")
colnames(tab1)=c("ID","Prediccion2")
colnames(tab2)=c("ID","Baja real")
join=full_join(tab1,tab2,by="ID")
tabla=table(join[,2],join[,3],useNA = "always");tabla

# Curva ROC y valor AUC:
Pred_roc <- predict(Arbol,dat_P, type="prob")
pred <- prediction(Pred_roc[,1],dat_P$Baja.objetivo,label.ordering = c("SI","NO"))
perf <- performance(pred,"tpr","fpr")
plot(perf,col="red",main="Curva ROC")
abline(a=0,b=1,main="Curva ROC")
performance(pred,"auc")

# Otra opción: Duplicando la clase minoritaria y reduciendo al 50%
# la clase mayoritaria

# Duplicamos los SI:
datos_rep1=rbind(dat_NP_SI,dat_NP_SI)      #duplicamos

# Reducimos al 50% los NO:

n1=32561 #50% de los NO
n=length(dat_NP_NO[,1]);n
i=1:n
ii=sample(i,size=n1,replace=FALSE)      #reduccion(sorteo)
datos_rep2=dat_NP_NO[ii,]
datos_rep=rbind(datos_rep1,datos_rep2)  #unimos los SI con los NO

datos_rep=sample(datos_rep)             #mezclamos los SI y los NO

# Modelo
Arbol<-J48(eval(parse(text=formula)),data=datos_rep)

# Predicción
Prediccion <- predict(Arbol,dat_P, type="class")
```

```
# Matriz de confusion(validacion)
tabla=table(Prediccion,dat_P$Baja.objetivo);tabla

# Matriz de confusión
confusionMatrix(tabla,positive ="SI")

# Nº de predicciones
Prediccion2 <- predict(Arbol,prediccion,type="class")
table(Prediccion2)

# Tabla a futuro (con los NA)
training=read.csv2("training_fijo_junio.csv",header = TRUE)

tab1=data.frame(prediccion$Producto.base_ID.Producto,Prediccion2)
tab2=data.frame(training$Producto.base_ID.Producto,training$Baja.objetivo) #training junio



library("dplyr")
colnames(tab1)=c("ID","Prediccion2")
colnames(tab2)=c("ID","Baja real")
join=full_join(tab1,tab2,by="ID")
tabla=table(join[,2],join[,3],useNA = "always");tabla

# Curva ROC y valor AUC:
Pred_roc <- predict(Arbol,dat_P, type="prob")
pred <- prediction(Pred_roc[,1],dat_P$Baja.objetivo,label.ordering = c("SI","NO"))
perf <- performance(pred,"tpr","fpr")
plot(perf,col="red",main="Curva ROC")
abline(a=0,b=1,main="Curva ROC")
performance(pred,"auc")

#############
# BOOSTING #
#############

library(adabag)  #librería para el boosting

# Modelo:

adaboost <- boosting(eval(parse(text=formula)), data=dat_NP, boos=TRUE, mfinal=2)

# Resultados:
Prediccion=adaboost$class

# Predicciones
tabla=table(Prediccion,dat_NP$Baja.objetivo);tabla    #Matriz de confusion
confusionMatrix(tabla,positive ="SI")

# Número de predicciones:
Prediccion2<-predict(adaboost,prediccion,type="class")$class
table(Prediccion2)
```

```
# Matriz a futuro:
training=read.csv2("training_fijo_junio.csv",header = TRUE)

tab1=data.frame(prediccion$Producto.base_ID.Producto,Prediccion2)
tab2=data.frame(training$Producto.base_ID.Producto,training$Baja.objetivo)

library("dplyr")
colnames(tab1)=c("ID","Prediccion2")
colnames(tab2)=c("ID","Baja real")
join=full_join(tab1,tab2,by="ID")
tabla=table(join[,2],join[,3],useNA = "always");tabla

# Curva ROC y valor AUC:
library(ROCR)
Pred_roc <- predict(adaboost,dat_P, type="prob")$votes
pred <- prediction(Pred_roc[,1],dat_P$Baja.objetivo,label.ordering = c("SI","NO"))
perf <- performance(pred,"tpr","fpr")
plot(perf,col="red",main="Curva ROC")
abline(a=0,b=1,main="Curva ROC")
performance(pred,"auc")

####################
## Random Forest ##
####################

# CASO A: datos brutos

library(randomForest)

training2=training[,-c(1,65,66,67,68,69,70,71)]
formula=paste("Baja.objetivo", "~",paste(colnames(training2)[1:(ncol(training2)-1)],
collapse="+"))

# Modelo predictivo: 8 variables y 5 árboles a combinar (con reemplazamiento)
Modelo <- randomForest(eval(parse(text=formula)),data=training2,ntree=5,mtry=8,replace=T)

# Predicciones:
Prediccion <- predict(Modelo,dat_P)

# Matriz de confusión:
MC<-table(Prediccion,dat_P$Baja.objetivo)
MC

library(caret)
confusionMatrix(MC,positive="SI")

# Nº de predicciones
Prediccion2 <- predict(Modelo,prediccion)
table(Prediccion2)
```

```
# Matriz a futuro:
training=read.csv2("training_fijo_junio.csv",header = TRUE)

tab1=data.frame(prediccion$Producto.base_ID.Producto,Prediccion2)
tab2=data.frame(training$Producto.base_ID.Producto,training$Baja.objetivo) #training junio

library("dplyr")
colnames(tab1)=c("ID","Prediccion2")
colnames(tab2)=c("ID","Baja real")
join=full_join(tab1,tab2,by="ID")
tabla=table(join[,2],join[,3],useNA = "always");tabla

# Curva ROC:

library(ROCR)
Pred_roc <- predict(Modelo,dat_P, type="prob")
pred <- prediction(Pred_roc[,1],dat_P$Baja.objetivo,label.ordering = c("SI","NO"))
perf <- performance(pred,"tpr","fpr")
plot(perf,col="red",main="Curva ROC")
abline(a=0,b=1,main="Curva ROC")
performance(pred,"auc")

# CASO B: Random Forest dividiendo entre permite y no permite

dat_NP=subset(training,training$Cuentas_Acc..Llamadas=="NO PERMITE")
dat_P=subset(training,training$Cuentas_Acc..Llamadas=="PERMITE")

library(randomForest)

dat_NP2=dat_NP[,-c(1,65,66,67,68,69,70,71)] #seleccion sobre la muestra NO PERMITE
formula=paste("Baja.objetivo", "~",paste(colnames(dat_NP2)[1:(ncol(dat_NP2)-1)],collapse="+"))

# Modelo predictivo:
 # Seleccionamos primero el numero de variables:
 n <- ceiling(sqrt(length(dat_NP2[2,])))
 # Aplicamos el modelo:
 Modelo <- randomForest(eval(parse(text=formula)),data=dat_NP2,ntree=60,mtry=n,replace=T)

# Predicciones:
Prediccion <- predict(Modelo,dat_P)

# Matriz de confusion:
MC<-table(Prediccion,dat_P$Baja.objetivo)
MC

library(caret)
confusionMatrix(MC,positive="SI")

# Nº de predicciones:
Prediccion2 <- predict(Modelo,prediccion)
table(Prediccion2)
```

```
# Matriz a futuro:
training=read.csv2("training_fijo_junio.csv",header = TRUE)

tab1=data.frame(prediccion$Producto.base_ID.Producto,Prediccion2)
tab2=data.frame(training$Producto.base_ID.Producto,training$Baja.objetivo) #training junio
library("dplyr")
colnames(tab1)=c("ID","Prediccion2")
colnames(tab2)=c("ID","Baja real")
join=full_join(tab1,tab2,by="ID")
tabla=table(join[,2],join[,3],useNA = "always");tabla

# Curva ROC:

library(ROCR)
Pred_roc <- predict(Modelo,dat_P, type="prob")
pred <- prediction(Pred_roc[,1],dat_P$Baja.objetivo,label.ordering = c("SI","NO"))
perf <- performance(pred,"tpr","fpr")
plot(perf,col="red",main="Curva ROC")
abline(a=0,b=1,main="Curva ROC")
performance(pred,"auc")

# Visualización gráfica de los árboles:
plot(Modelo,type="l")

#Valores de la gráfica del error para cada árbol construido:
Modelo$err.rate

###################
#### BAGGING  ####
###################

library(adabag)

#Modelo predictivo:

Modelo <- bagging(eval(parse(text=formula)),data=dat_NP,mfinal=2)     #2 arboles

# Predicciones:

Prediccion <- predict(Modelo,dat_P,type="class")$class

# N° Predicciones:

Prediccion2 <- predict(Modelo,prediccion,type="class")$class

# Matriz de confusion:

MC<-table(Prediccion,dat_P$Baja.objetivo);MC
confusionMatrix(MC,positive="SI")
```

```
# Matriz a futuro:
training=read.csv2("training_fijo_junio.csv",header = TRUE)

tab1=data.frame(prediccion$Producto.base_ID.Producto,Prediccion2)
tab2=data.frame(training$Producto.base_ID.Producto,training$Baja.objetivo) #training junio

library("dplyr")
colnames(tab1)=c("ID","Prediccion2")
colnames(tab2)=c("ID","Baja real")
join=full_join(tab1,tab2,by="ID")
tabla=table(join[,2],join[,3],useNA = "always");tabla

# Curva ROC:
Pred_roc <- predict(Modelo,dat_P, type="prob")$votes
pred <- prediction(Pred_roc[,1],dat_P$Baja.objetivo,label.ordering = c("SI","NO"))
perf <- performance(pred,"tpr","fpr")
plot(perf,col="red",main="Curva ROC")
abline(a=0,b=1,main="Curva ROC")
performance(pred,"auc")

 ############################
 ### MÉTODOS AdaOUBoost ###
 ############################

 ### K=2 clusters ###

 # Undersampling(k-means)
 ind=which(sapply(dat_NP_NO,is.numeric))
 dat_num=dat_NP_NO[,ind]
 cluster=kmeans(dat_num,centers=2)        #clustering: k=2 clusters

 ind1=which(cluster$cluster==1);N1=dat_NP_NO[ind1,] #cluster 1
 ind2=which(cluster$cluster==2);N2=dat_NP_NO[ind2,] #cluster 2

 # Oversampling(SMOTE):

 library(caret)
 library(DMwR)

 train1=rbind(dat_NP_SI,N1);train1=sample(train1)
 train2=rbind(dat_NP_SI,N2);train2=sample(train2)

 smote1=SMOTE(Baja.objetivo~. ,data=train1,perc.over=2000,perc.under=200)
 smote2=SMOTE(Baja.objetivo~. ,data=train2,perc.over=2000,perc.under=200)

 # Árbol de decisión para cada subconjunto:

 #Primer subconjunto:
```

```
#Modelo:
formula=paste("Baja.objetivo", "~",paste(colnames(training)[2:71],collapse="+"))
Arbol1=J48(eval(parse(text=formula)),dat=smote1)

#Predicciones:
Prediccion1 <- predict(Arbol1,dat_P, type="class")
tabla1=table(Prediccion1,dat_P$Baja.objetivo);tabla1

#Métricas y matriz de clasificación:
confusionMatrix(tabla1,positive="SI")

#Número de predicciones:
Predict1 <- predict(Arbol1,prediccion, type="class")
table(Predict1)

#Segundo subconjunto:

#Modelo:
formula=paste("Baja.objetivo", "~",paste(colnames(training)[2:71],collapse="+"))
Arbol2=J48(eval(parse(text=formula)),dat=smote2)

#Predicciones:
Prediccion2 <- predict(Arbol2,dat_P, type="class")    #Prediccción en Test
tabla2=table(Prediccion2,dat_P$Baja.objetivo);tabla2

#Métricas y matriz de clasificación:
confusionMatrix(tabla2,positive="SI")

#Numero de predicciones:
Predict2 <- predict(Arbol2,prediccion, type="class")
table(Predict2)

# Arbol final:

# Predicciones de las prob. de clasif. de cada individuo para cada arbol:
Pred1 <- predict(Arbol1,dat_P, type="prob")
Pred2 <- predict(Arbol2,dat_P, type="prob")

r=length(Pred1[,2])
# Media ponderada de las prob. de cada arbol para cada cliente
Pred=numeric(r)
for (i in 1:r){
Pred[i]=mean(c(Pred1[i,2],Pred2[i,2]))
}

Pred_final=numeric(r)
for (i in 1:r){
if (Pred[i]>0.5){Pred_final[i]="SI"}
else{Pred_final[i]="NO"}
}
Pred_final=as.factor(Pred_final)
```

```
# Matriz de confusión:
tabla_final=table(Pred_final,dat_P$Baja.objetivo)
confusionMatrix(tabla_final,positive="SI")

### Construcción del PRED:

n=length(Pred1[,1])
PRED=matrix(0,nr=n,nc=2)
for (i in 1:n){
PRED[i,1]=mean(c(Pred1[i,1],Pred2[i,1]))
PRED[i,2]=mean(c(Pred1[i,2],Pred2[i,2]))
}

# N° de predicciones:
npred1 <- as.numeric(table(Predict1)[2])
npred2 <- as.numeric(table(Predict2)[2])
npred <- floor(mean(c(npred1,npred2)));npred

# Matriz a futuro:(calcular Prediccion2 promediando antes)
training=read.csv2("training_fijo_junio.csv",header = TRUE)

tab11=data.frame(prediccion$Producto.base_ID.Producto,Predict1)
tab12=data.frame(prediccion$Producto.base_ID.Producto,Predict2)
tab2=data.frame(training$Producto.base_ID.Producto,training$Baja.objetivo)

library("dplyr")
colnames(tab11)=c("ID","Prediccion");colnames(tab12)=c("ID","Prediccion")
colnames(tab2)=c("ID","Baja real")

join1=full_join(tab11,tab2,by="ID")
tabla1=table(join1[,2],join1[,3],useNA = "always")

# Curva ROC:

library(ROCR)
pred <- prediction(PRED[,1],dat_P$Baja.objetivo,label.ordering = c("SI","NO"))
perf <- performance(pred,"tpr","fpr")
plot(perf,col="red",main="Curva ROC")
abline(a=0,b=1,main="Curva ROC")
performance(pred,"auc")
```

Bibliografía

[1] Alejo Eleuterio, A (2010) Análisis del Error en Redes Neuronales: Corrección de los Datos y Distribuciones no balanceadas.

[2] Alfaro Cortés, E (2006) Combinación de clasificadores mediante el método boosting. Una aplicación a la predicción del fracaso empresarial en España. Universidad de Castilla la Mancha.

[3] Alfaro Cortés E, Gámez Martínez M, García Rubio N (2003) Una revisión de los métodos de agregación de clasificadores.

[4] Barrientos F, Ríos S (2013) Aplicación de minería de datos para predecir la fuga de clientes en la industria de las telecomunicaciones.

[5] Breiman Leo (1994) Bagging Predictors. Department of Statistics, University of California at Berkeley.

[6] Breiman Leo (2001) Random Forests. Statistics Department, University of California at Berkeley.

[7] Cao P, Zhao D, Zaiane O (2013) Hybrid probabilistic sampling with random subspace for imbalnced data learning.

[8] Carmona Francesc (2014) Un ejemplo de ACP paso a paso.

[9] Castro Casillas Gonzalo (2016) Uso de análisis asociativo en algoritmos de aprendizaje.

[10] Cateni S, Colla V, Vannucci M (2014) A method for resampling imbalanced datasets in binary classification tasks for real-world problems.

[11] Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) SMOTE: Synthetic Minority Over-Sampling Technique.

[12] De la Hoz Emiro, De la Hoz Eduardo M, Ortiz Andrés y Ortega Julio (2012) Modelo de detección de intrusiones en sistemas de red, realizando selección de características con FDR y entrenamiento y clasificación con SOM.

[13] Drummond C, Holte RC (2003) C4.5. Class Imbalance and Cost Sensitivity. Why Under-Sampling beats Over-Sampling.

[14] Fergus P, Cheung P, Hussain A, Al-Jumeily D, Dobbins C, Iram S (2013) Prediction of Preterm Deliveries from EHG Signals Usinf Machine Learning.

[15] Fernández Santana Óscar (1991) El Análisis Clúster: Aplicación, interpretación y validación.

[16] Flach P (2007) ROC analysis for ranking and probability estimation.

[17] Galar M, Fernández A, Barrenechea E, Bustince H, Herrera F (2011) A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches.

[18] Gálvez Carmen (2008) Minería de textos: la nueva generación de análisis de literatura científica en biología molecular y genómica.

[19] Garcia S, Herrera F (2009) Evolutionary Under-Sampling for Classification with Imbalanced Data Sets: Proposals and Taxonomy.

[20] García Jiménez María, Álvarez Sierra Aránzazu. Análisis de Datos en WEKA-Pruebas de Selectividad. Universidad de Carlos III, Ingeniería de la Telecomunicación.

[21] González WA (2013) Predicción de la evaluación hacia la hipertensión arterial en la adultez desde la adolescencia utilizando técnicas de aprendizaje automatizado.

[22] Goutte Cyril, Gaussier Eric (2005) A Probabilistic Interpretation of Precision, Recall and F-score, with Implication for Evaluation.

[23] Han H, Wang W-Y, Mao B-H (2005) Borderline-SMOTE: A new over-sampling method in Imbalanced data Sets Learning.

[24] Huang PJ (2015) Classification of Imbalanced Data Using Synthetic Over-Sampling Techniques.

[25] Infante Izquierdo Javier (2013) Análisis del comportamiento del CTC con problemas de clases muy desbalanceadas del repositorio KEEL.

[26] Japkowicz N (2000) The Class Imbalance Problem: Significance and Strategies.

[27] Kearns Michael (1988) Thoughts on Hypothesis Boosting.

[28] Kearns Michael, Valiant Leslie (1989) Crytographic limitations on learning Boolean formulae and finite automata.

[29] Korn Sue (2011) The Opportunity for Predictive Analytics in Finance.

[30] Liaw A, Wiener M (2002) Classification and Regression by randomForest.

[31] Liu X-Y, Wu J, Zhou Z-H (2009) Exploratory Undersampling for Class-Imbalance Learning.

[32] Loyola González O, Martínez Trinidad JF, García Barroto M (2014) Clasificadores supervisados basados en Patrones Emergentes para Bases de Datos con Clases Desbalanceadas.

[33] Luengo J, Fernández A, García S, Herrera F (2010) Addressing data complexity for Imbalanced data sets: analysis of SMOTE-based oversampling and evolutionary undersampling.

[34] Mason L, Baxter J, Bartlett PL, Frean M (1999) Boosting Algorithms as Gradient Descent in Fuction Space.

[35] Montero Pérez A, González Berbés E, Garijo Mazario FJ, Martín Nieto F, Gómez Verdejo V, Arenas García J, Navia Vázquez A, Figueiras Vidal AR (2004) Aplicaciones de aprendizaje no supervisado para la detección de patrones de fraude en telecomunicaciones.

[36] Moreno J, Rodriguez D, Sicilia MA, Riqueleme JC, Ruiz R (2009) SMOTE-I: mejora del algoritmo SMOTE para balanceo de clases minoritarias.

[37] Nadelsticher Abraham (2001) La técnica del Análisis CHAID.

[38] Peng Y, Yao J (2010) AdaOUBoost: Adaptative Over-Sampling and Under-Sampling to Boost the Concept Learning in Large Scale Imbalanced Data Set.

[39] Pita Fernández S, Pértega Díaz S (1997) Relación entre variables cuantitativas, Unidad de Epidemiología Clínica y Bioestadística. Complejo Hospitalario Juan Canalejo, A Coruña.

[40] Quinlan JR (1986) Induction of Decision Trees.

[41] Quinlan JR (1996) Improved Use of Continuous Attributes in C4.5.

[42] Rahman MM, Davis DN (2013) Cluster Based Under-Sampling for Unbalanced Cardiovascular Data.

[43] Rokach Lior and Maimon Oded (2008) Data mining with decision trees: theory and applications.

[44] Saar-Tsechansky Maytal, Provost Foster (2007) Handling Missing Values when Applying Classification Models.

[45] Salas Velasco Manuel (1996) La regresión logística. Una aplicación a la demanda de estudios universitarios.

[46] Schapire Robert E (1990) The Strength of Weak Learnability.

[47] Segrera Francia S, Moreno García MN (2006) Multiclasificadores: Métodos y Arquitecturas.

[48] Telgarsky M, Vattani A (2010) Hartigan's Method: k-means Clustering without Voronoi.

[49] Tomek I (1976) Two Modifications to CNN. IEEE Transactions Systems, Man, and Communications, 769-772.

[50] Vázquez Fernando, Salvador Sánchez J, Pla Filiberto (2005) A Stochastic Approach to Wilson?s Editing Algorithm.

[51] Wang Q (2014) A Hybrid sampling SVM Approach to Imbalanced Data Classification.

[52] Witten Ian H (2013) Data Mining with Weka.

[53] Yap BW, Abd Rani K, Abd Rahman HA, Frog S, Khairudin Z, Nairan Abdullah N (2013) An Application of Oversampling, Undersampling, Bagging and Boosting in Handling Imbalanced Datasets.

[54] Yen S-J, Lee Y-S, Lin C-H, Ying J-C (2008) Investigate the Effect of Sampling Methods for Imbalanced Data Distributions.

[55] Zhang H, Li M (2014) RWO-Sampling: A random walk over-sampling approach to imbalanced data classification.

Librerías

[56] Breiman L, Cutler A, Liaw A, Wiener M (2015) randomForest: Breiman and Cutler's Random Forests for Classification and Regression. R package version 4.6-12. https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

[57] Hornik K, Buchta C, Hothorn T, Karatzoglou A, Meyer D, Zeileis A (2016) RWeka: R/Weka Interface. R package version 0.4-29. https://cran.r-project.org/web/packages/RWeka/RWeka.pdf

[58] Hothorn T, Zeileis A (2016) partykit: A Toolkit for Recursive Partytioning. R package version 1.1-1. https://cran.r-project.org/web/packages/partykit/partykit.pdf

[59] Kuhn M (2016) caret: Classification and Regression Training. R package version 6.0-73. ftp://cran.r-project.org/pub/R/web/packages/caret/caret.pdf

[60] Milborrow Stephen (2016) rpart.plot: Plot 'rpart' Models: An Enhanced Version of 'plot.rpart'. R package version 2.1.0. https://cran.r-project.org/web/packages/rpart.plot/rpart.plot.pdf

[61] Sing T, Sander O, Beerenwinkel N, Lengauer T (2015) ROCR: Visualizing the Performance of Scoring Classifiers. R package versión 1.0-7. https://cran.r-project.org/web/packages/ROCR/ROCR.pdf

[62] Therneau T, Atkinson B, Ripley B (2015) rpart: Recursive Partitioning and Regression Trees. R package version 4.1-10. https://cran.r-project.org/web/packages/rpart/rpart.pdf

[63] Torgo L (2013) DMwR: Functions and data for "Data Mining with R". R package version 0.4.1. https://cran.r-project.org/web/packages/DMwR/DMwR.pdf

[64] Wickham H, Francois R (2016) dplyr: A Grammar of Data Manipulation. R package version 0.5.0. https://cran.r-project.org/web/packages/dplyr/dplyr.pdf