Herramientas del BIG DATA y analytics

Una introducción a los modelos de Clasificación empleando R

Autores:
Julio César Alonso
Cristian Camilo Hoyos

Universidad Icesi | Editorial Universidad Icesi

Una introducción a los modelos de Clasificación empleando R

Julio César Alonso[1] Cristian Camilo Hoyos[2]

2025-03-04

[1]Departamento de Economía - Universidad Icesi, jcalonso@icesi.edu.co
[2]Departamento de Economía - Universidad Icesi

© **Una introducción a los modelos de Clasificación empleando R**

Julio César Alonso C. - Cristian Camilo Hoyos B.

Colección «Herramientas del Big Data y Analytics», vol. 6

Cali. Universidad Icesi, 2025.

185 páginas.

Incluye referencias bibliográficas.

ISBN: 978-628-7740-98-3 (eBook).

DOI: https://doi.org/10.18046/EUI/bda.h.5

Palabras Clave: 1. R | 2. Analítica | 3. Modelos de clasificación | 4. Métricas de valoración | 5. Big Data Analytics

Clasificación Dewey: 545 ddc 21

© **Universidad Icesi**

CIENFI - Centro de Investigación en Economía y Finanzas

www.icesi.edu.co/centros-academicos/cienfi

Rector: Esteban Piedrahita Uribe

Secretaria General: Olga Patricia Ramírez Restrepo

Director Académico: José Hernando Bahamón

Coordinador editorial: Adolfo A. Abadía

Corrección de estilo: Sandra M. Cubillos G.

Diseño de portada: Sandra Moreno

Fotos tomadas por: Julio César Alonso

Editorial Universidad Icesi

Calle 18 No. 122-135 (Pance), Cali – Colombia

Teléfono: +57 (2) 555 2334 | E-mail: editorial@icesi.edu.co

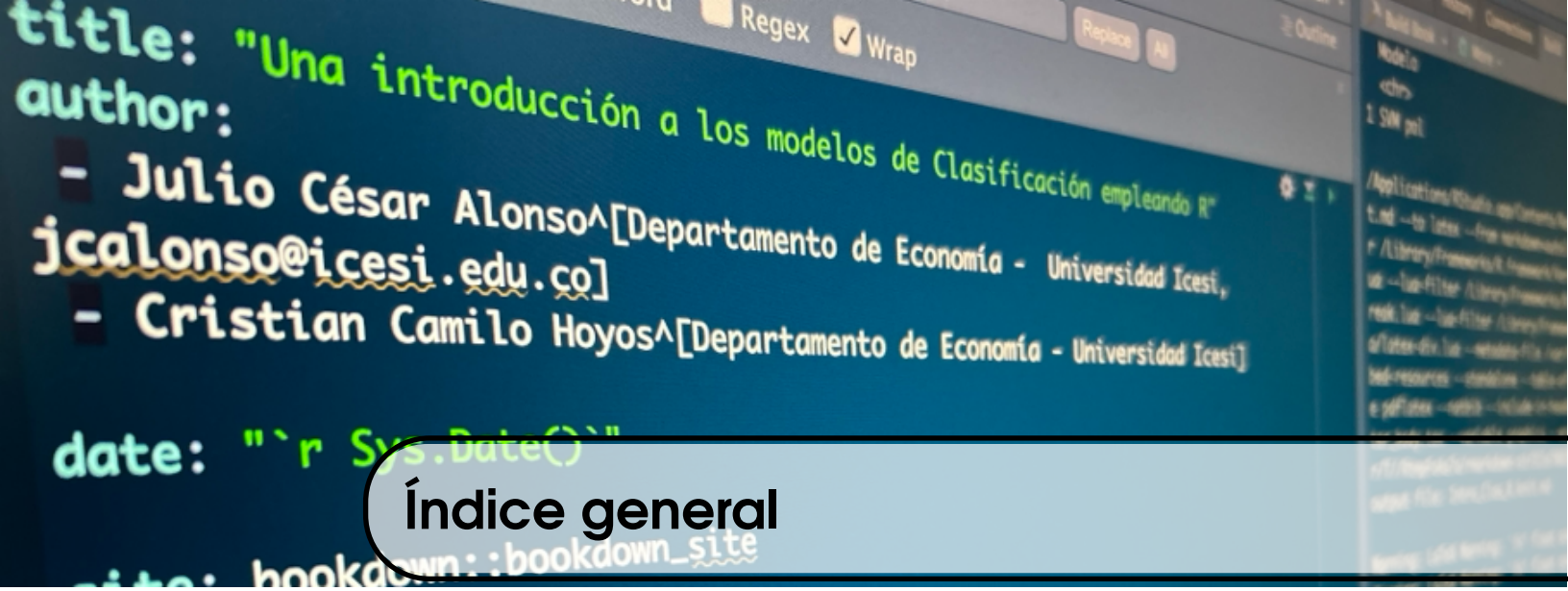
http://www.icesi.edu.co/editorial

Publicado en Colombia – *Published in Colombia*

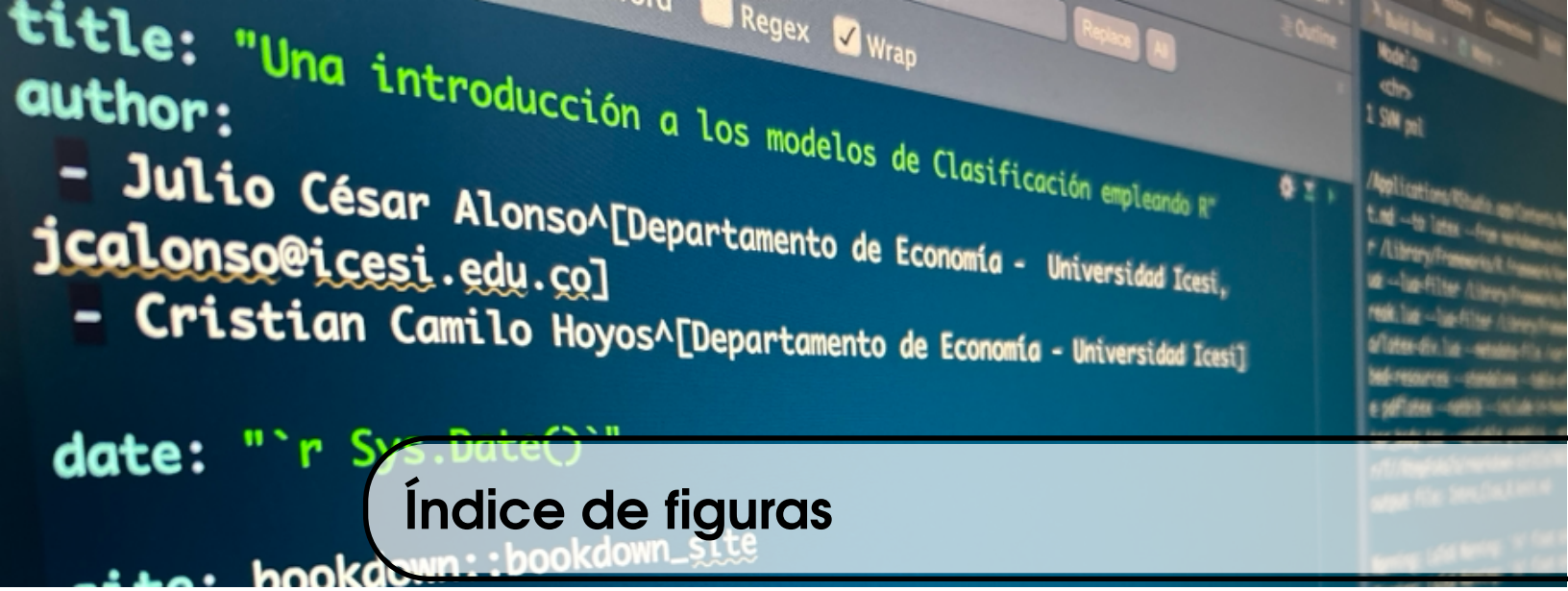
La publicación de este libro se aprobó luego de superar un proceso de evaluación doble ciego.

La Editorial Universidad Icesi no se hace responsable de las ideas expuestas bajo su nombre, las ideas publicadas, los modelos teóricos expuestos o los nombres aludidos por los autores. El contenido publicado es responsabilidad exclusiva de los autores, no refleja la opinión de las directivas, el pensamiento institucional de la Universidad Icesi, ni genera responsabilidad frente a terceros en caso de omisiones o errores.

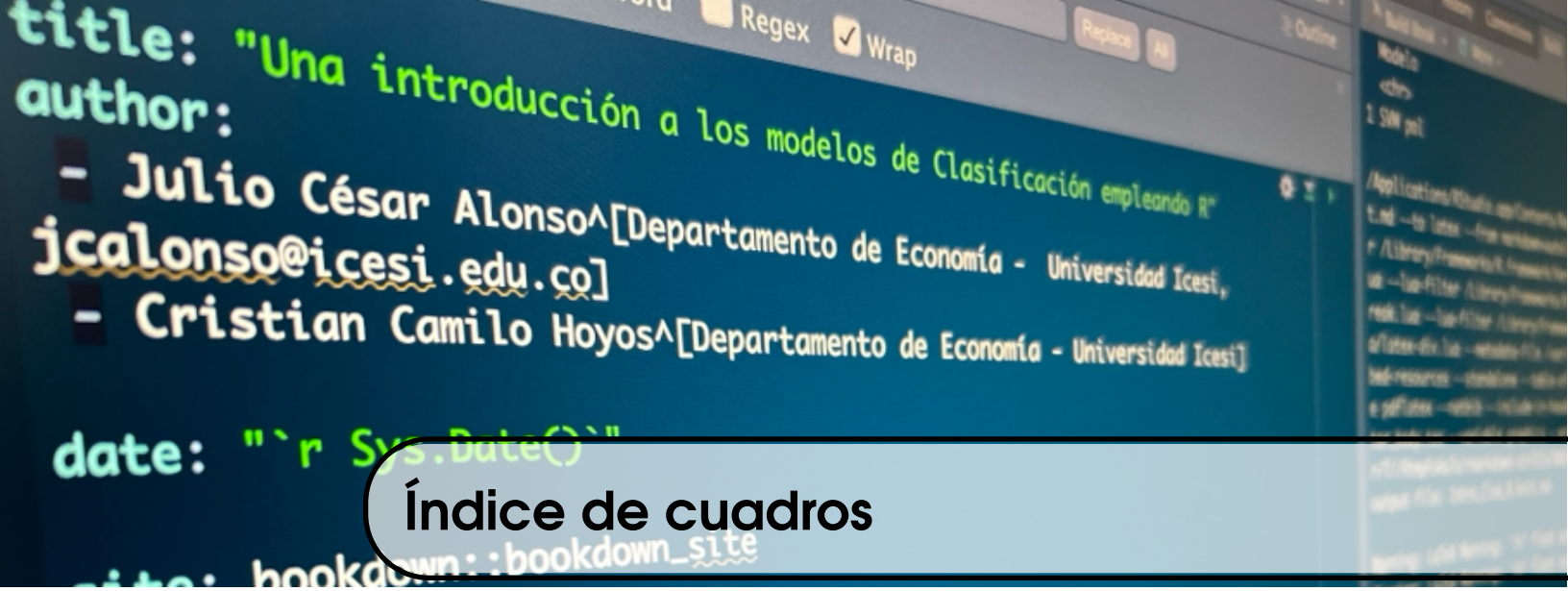
El material de esta publicación puede ser reproducido sin autorización, siempre y cuando se cite título, autor(es) y fuente institucional.



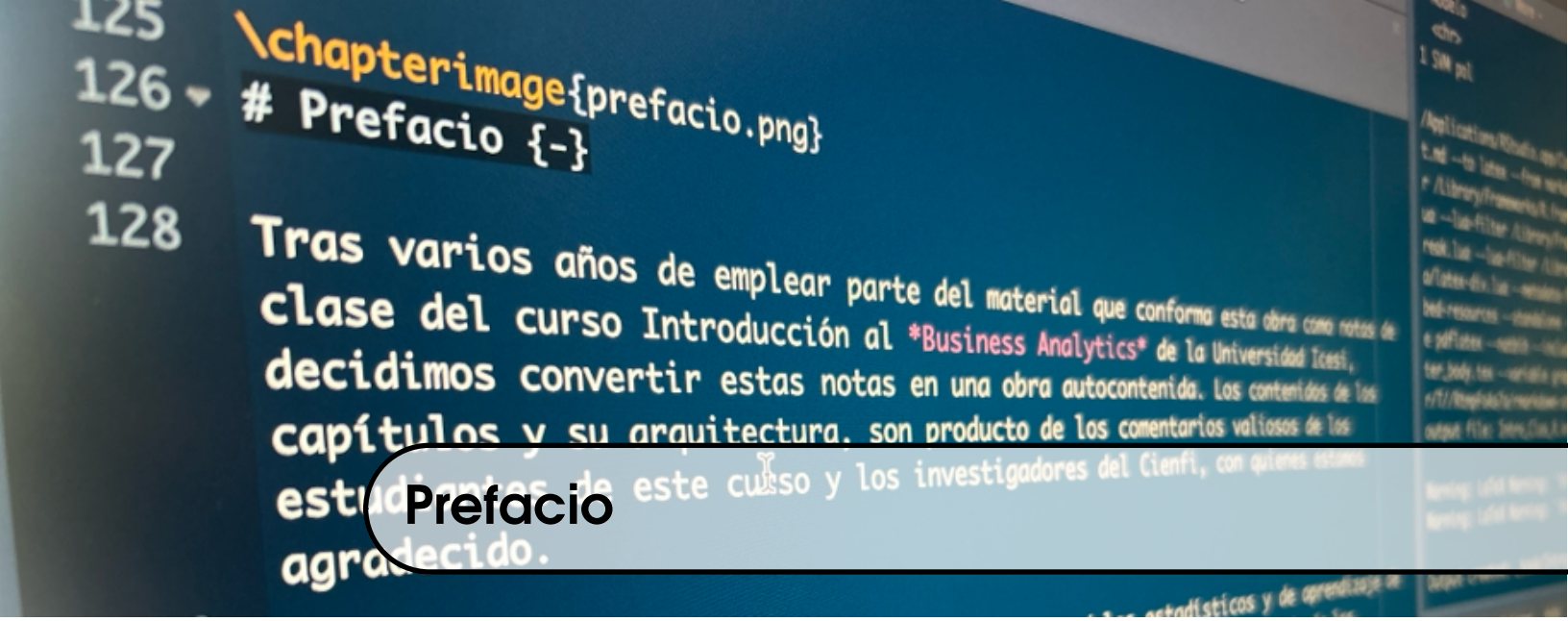
Índice general



Índice de figuras



Índice de cuadros



Prefacio

Tras varios años de emplear parte del material que conforma esta obra como notas de clase del curso Introducción al *Business Analytics* de la Universidad Icesi, decidimos convertir estas notas en una obra autocontenida. Los contenidos de los capítulos y su arquitectura, son producto de los comentarios valiosos de los estudiantes de este curso y los investigadores del Cienfi, con quienes estamos agradecido.

Este libro presenta una introducción a los modelos estadísticos y de aprendizaje de máquina que permiten realizar la tarea de clasificación. La discusión de los diferentes capítulos está dirigida a personas que están empezando su formación de científico de datos.

Este libro supone un uso intermedio de R (R Core Team, 2023). Si crees que necesitas algún refuerzo en R, recomendamos tres libros. Alonso y Ocampo (2022) presenta una breve introducción para iniciar a usar R. Ese primer libro discute cómo instalar R y RStudio y paquetes, cómo cargar diferentes bases de datos y cómo realizar operaciones aritméticas y lógicas con objetos. En Alonso y Ocampo (2022) también se discuten las clases esenciales de objetos sencillos y compuestos. No dudes en consultar ese primer libro si aún no has iniciado tu camino por el universo de R.

El segundo libro de la serie (Alonso, 2022) presenta una breve introducción al paquete para *dplyr* (Wickham et al., 2021) que permite manipular objetos que contengan datos. En ese libro se discute cómo filtrar observaciones, crear nuevas variables y combinar objetos con datos. Es recomendable tener un conocimiento de ese paquete antes de leer esta obra. Consulta ese segundo libro si aún no has tenido alguna experiencia manipulando objetos con datos con *dplyr*.

Finalmente, recomendamos (Alonso y Largo, 2023) en el que se presenta una introducción a la creación de visualizaciones con el paquete *ggplot2* (Wickham, 2016). En esta obra emplearemos visualizaciones empleando este paquete. Así este libro asume un manejo intermedio de R, y los paquetes *dplyr* y *ggplot2*.

Por otro lado, un manejo del modelo de regresión múltiple conceptualmente y en

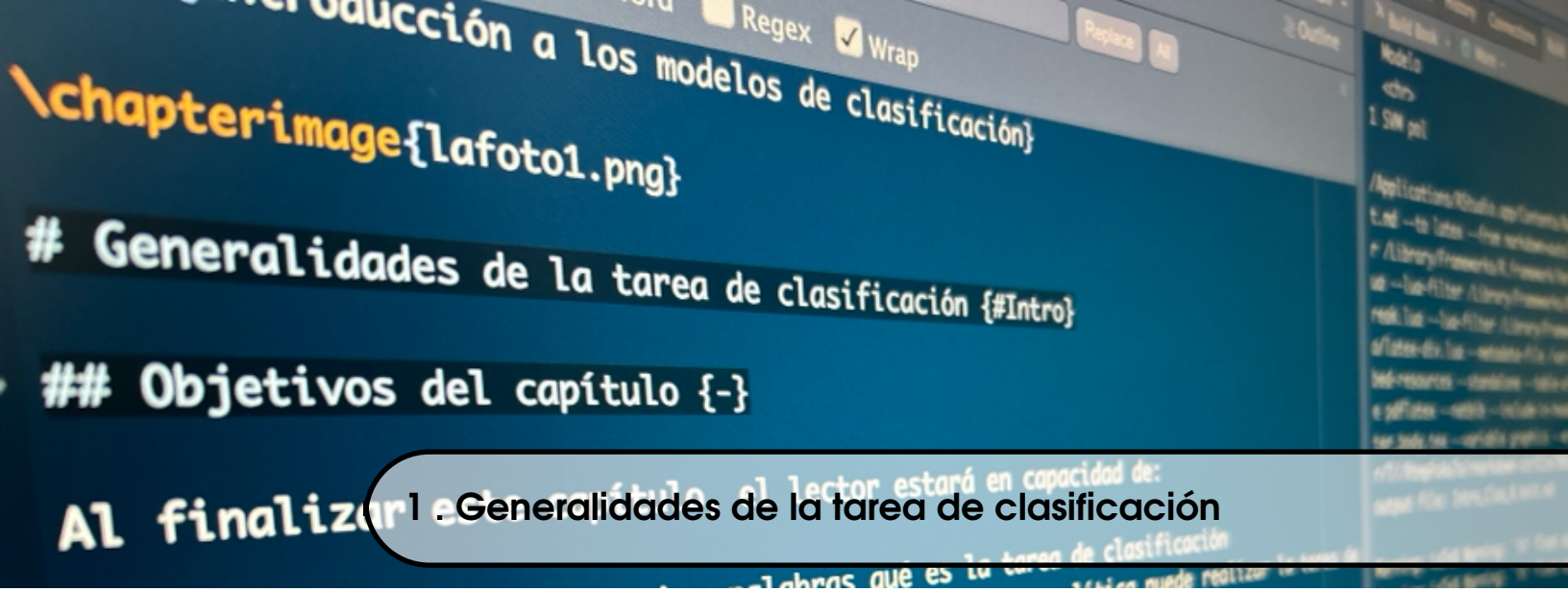
R es deseable. Alonso (2024) te puede brindar una introducción a la fundamentación formal del modelo de regresión y cómo estimar estos modelos y chequear sus supuestos en R.

Esta obra recoge nuestra experiencia trabajando con R y los modelos de clasificación para resolver problemas con datos desde el Cienfi (Centro de Investigación en Economía y Finanzas) de la Universidad Icesi. En el Cienfi, empleamos R para la transformación de datos en conclusiones que faciliten la toma de decisiones en organizaciones privadas y públicas.

¡Esperamos encuentres esta obra útil y la compartas con otros! Si tienes alguna sugerencia del libro o corrección, no dudes en escribirnos. Esta es una obra en constante construcción.

Parte I

Introducción a los modelos de clasificación



1 . Generalidades de la tarea de clasificación

Objetivos del capítulo

Al finalizar este capítulo, el lector estará en capacidad de:

- Explicar en sus propias palabras qué es la tarea de clasificación
- Explicar en sus propias palabras qué tipos de analítica puede realizar la tarea de clasificación

1.1 Introducción

En los últimos años se ha acelerado la transformación del mundo de los negocios por la disponibilidad de mayores volúmenes de datos y capacidad de computo que permiten emplear modelos de analítica para responder preguntas de negocio y así generar valor a las organizaciones (Alonso, 2024). Emplear herramientas de *Business Analytics* y *Big Data* permiten tanto optimizar los procesos actuales como generar características diferenciadoras para cada organización.

Los datos se han convertido en un recurso estratégico para las organizaciones, y el *business analitycs* es la herramienta mediante la cual, las organizaciones pueden monetizar ese recurso (Alonso, 2024). El *business analitycs* es el proceso científico de transformar datos en *insights* con el propósito de tomar mejores decisiones (Alonso, 2024).

Las tareas del *business analitycs* implican la necesidad de escoger la mejor herramienta para responder la pregunta de negocio planteada. Las respuestas a las preguntas de negocio implican diferentes tareas que se pueden clasificar en una de las siguientes categorías:

- *Resumir*
- *Visualizar*
- *Clusterizar* (*Agrupar*)
- *Clasificar*
- *Detectar excepciones*
- *Encontrar reglas de asociación* (*o buscar coocurrencia de productos*)[1]
- *Pronosticar*
- *Estimar regresiones*[2]

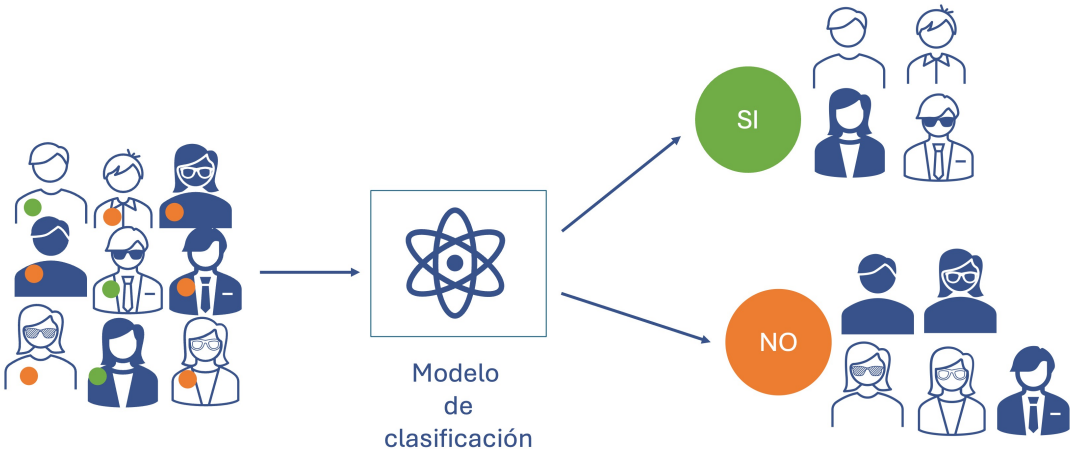
La tarea de *Clasificación* tiene como finalidad construir un modelo que permita predecir la categoría de un individuo o un objeto. En otras palabras, se emplea para categorizar o clasificar datos en diferentes grupos o clases. Esta tarea es útil para analizar conjuntos de datos para los cuáles queremos predecir la pertenencia a una categoría específica en función de ciertas características o atributos de los individuos.

Por ejemplo, en algunas situaciones se deseará determinar si un nuevo cliente comprará o no nuestro producto. En este caso, las categorías son *compra* o *no compra*. En la Figura 1.1 se presenta una representación gráfica de esta tarea.

[1] Para una discusión detallada de esta tarea y cómo implementarla en R se puede consular Alonso y Arboleda (2025).

[2] Para una discusión detallada del modelo clásico de regresión y cómo implementarlo en R se puede consultar Alonso (2024).

Figura 1.1. Tarea de clasificación



Fuente: elaboración propia.

Figura 1.2. Material multimedia: tarea de clasificación

Fuente: elaboración propia. https://youtu.be/0K7ryP0uKGo

Más ejemplos de preguntas de negocio que se pueden responder con la tarea de clasificación son:

- ¿Cuáles clientes cancelarán su suscripción a nuestro servicio?
- ¿Qué clientes no pagarán sus facturas a tiempo?
- ¿Cuáles de los candidatos serán un buen "match" para la posición que está disponible?
- ¿Qué productos seran devueltos por los clientes?
- ¿Cuáles de los proyectos se completarán dentro del plazo establecido?
- ¿Qué clientes responderán comprando el producto tras recibir una oferta de descuento?
- ¿Qué clientes comprarán un producto complementario?
- ¿Cuáles *leads*[3] se convertirán en clientes?
- ¿Qué clientes recomendarán nuestros productos o servicios a otros?

Como ya te debe quedar claro, la gama de preguntas de negocio que se pueden responder con la tarea de clasificación es muy grande.

Por otro lado, aunque suene raro, es importante anotar que la tarea de clasificación se realiza típicamete con modelos de clasificación como el modelo *Logit* (Ver Capítulo 3), el modelo clasificador bayesiano ingenuo o *Naive Bayes* en inglés (Ver Capítulo 5), el modelo **kNN** (Ver Capítulo 6), Árboles de decisión (Ver Capítulo 7), el modelo *Random Forest* (Ver Capítulo 8) y *Suport Vector Machine* o **SVM** (Ver Capítulo 9), entre otros. Esos modelos, en algunas ocasiones pueden ser empleados para hacer otro tipo de tarea de analítica. Por ejemplo, algunos modelos de clasificación pueden ser empleados para detectar excepciones; y más específico anomalías. Por ejemplo se puede emplear modelos de clasificación para responder la pregunta ¿es esta operación de tarjeta de crédito fraudulenta o no? Así, la tarea de clasificación se realiza con modelos de clasificación, sin embargo, los modelos de analítica pueden servir para hacer tareas diferentes a la clasificación.

Los modelos de clasificación se estiman o entrenan empleando una muestra de individuos[4] para los cuales se cuenta con observaciones de las características de estos y su clasificación (variable objetivo). Así estos modelos corresponden a modelos de aprendizaje supervisado[5], pues la muestra contiene la variable objetivo que se encuentra etiquetada con la característica bajo estudio[6]. La misión del modelo o algoritmo es aprender de esos datos cuál es el patrón que permitiría predecir la categoría (variable objetivo) de un individuo dadas las características de este.

Regresando a las tareas del *business analytics*, una manera más común de clasificar los ejercicios de analítica es según su propósito. En ese caso se tienen cuatro tipos de

[3] En este contexto, los *leads* son los posibles clientes potenciales que han mostrado interés en el producto o servicio de una empresa.

[4] Típicamente para los modelos de clasificación se emplean muestras de corte transversal. Es decir, a muchos individuos se les observa sus características y la variable objetivo en el mismo período.

[5] En el campo del aprendizaje de máquina se distinguen dos tipos de aprendizaje: supervisado y no supervisado. En el aprendizaje no supervisado los datos no contienen etiquetas o la "respuesta correcta". El aprendizaje no supervisado busca descubrir patrones o estructuras ocultas en los datos.

[6] Es decir, con la respuesta correcta ya conocida.

analítica (Ver Figura 1.3):

- **Analítica Descriptiva**: Esta analítica se enfoca en resumir y visualizar los datos para obtener información sobre lo que ha sucedido en el pasado. Ayuda a comprender patrones y tendencias.
- **Analítica Diagnóstica**: Esta analítica busca entender por qué algo ha sucedido. Examina los datos para identificar las causas raíz de los problemas o éxitos pasados.
- **Analítica Predictiva**: Esta analítica utiliza modelos estadísticos y de aprendizaje de máquina para hacer pronósticos y predecir eventos futuros.
- **Analítica Prescriptiva**: Esta analítica se centra en recomendar acciones y soluciones óptimas para lograr un objetivo.

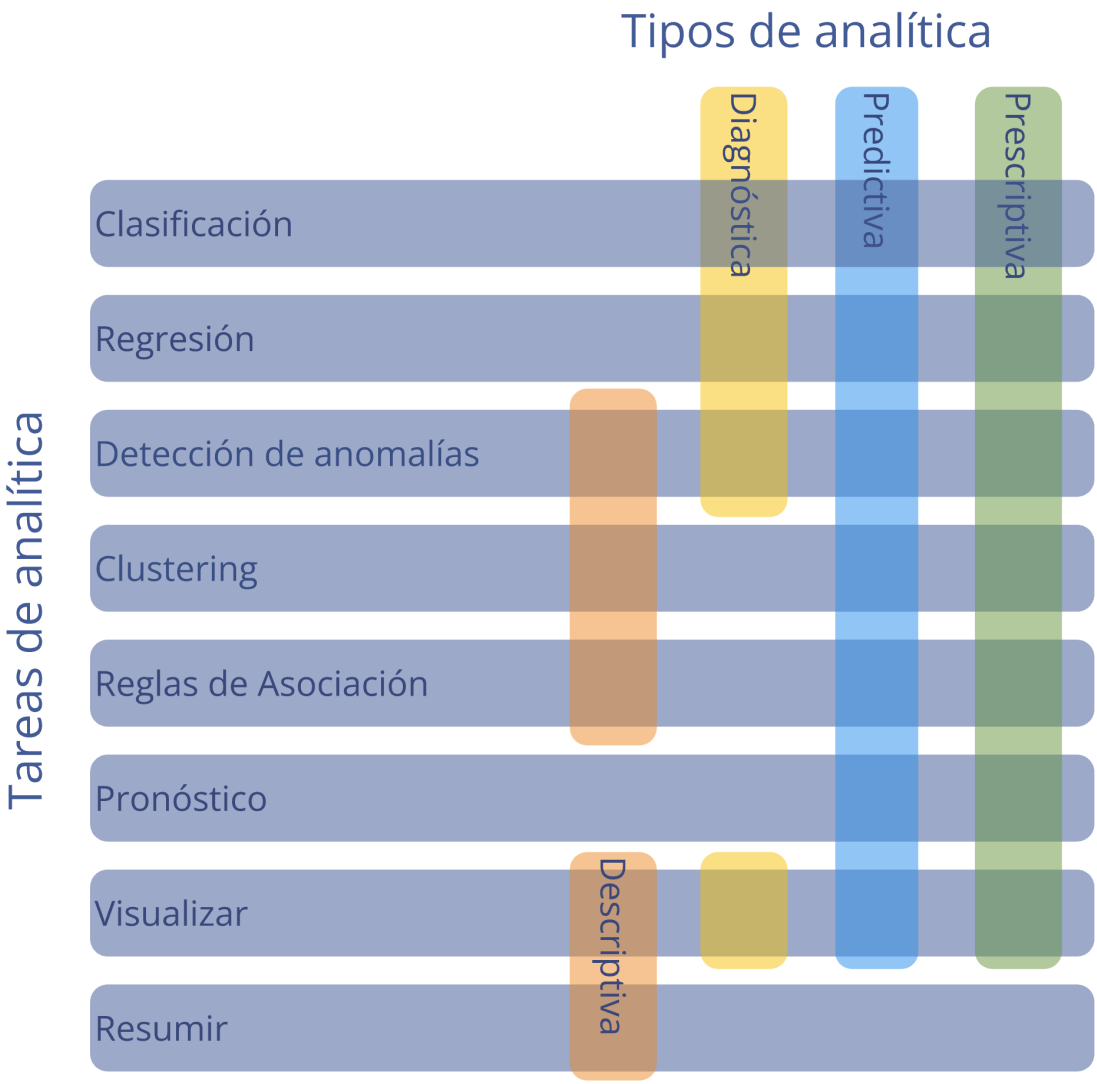
Figura 1.3. Material multimedia: tipos de analítica

Fuente: elaboración propia. https://rb.gy/s7efwr

Estos cuatro tipos de analítica engloban las ocho tareas del *business analytics*. La tarea de clasificación permite realizar analítica diagnóstica, descriptiva y prescriptiva (Ver Figura 1.4).

La **analítica diagnóstica** responde a la pregunta **¿por qué está pasando algo?** En el contexto de las preguntas de clasificación, dependiendo de los modelos que se empleen, se podrá determinar por qué un individuo queda clasificado en una categoría u otra. Por ejemplo, supongamos que queremos responder la pregunta de negocio ¿qué variables son las más importantes para que un cliente siga siendo mi cliente en los próximos 6 meses? Esta pregunta implica una tarea de clasificación: clasificar a los clientes actuales entre aquellos que continuarán después de 6 meses y aquellos que no, dadas sus características. Los modelos de clasificación (como el modelo *Logit* que será discutido en el Capítulo 3) pueden determinar cuáles variables son importantes al momento de determinar si un cliente dejará de ser mi cliente y cuáles no. Y para

Figura 1.4. Relación entre las tareas de analítica y los tipos de analítica



Fuente: elaboración propia.

un determinado individuo podremos saber el porqué de su no continuidad; es decir, cuáles variables pesaron más en ese individuo para no continuar como cliente.

La **analítica predictiva** busca responder la pregunta: **¿Qué es posible que ocurra?**. Los modelos de clasificación podrán determinar que categoría probablemente tendrá un individuo dadas sus características. Por ejemplo, supongamos que una institución del sector financiero quiere responder la pregunta de negocio ¿me pagará este individuo si le otorgo un crédito de libre inversión? En ese caso, un modelo de clasificación podrá clasificar a un individuo, dadas sus características, entre las categorías si pagará y no pagará. Así, el modelo permite predecir el comportamiento que tendrá ese individuo.

La **analítica prescriptiva** busca responder la pregunta: **¿qué necesito hacer?** Los modelos de clasificación nos permiten hacer este tipo de analítica al sugerir qué deberíamos hacer en algunas situaciones. Por ejemplo, un negocio quisiera responder a la pregunta ¿cuáles variables accionables[7] puedo manipular para lograr que un cliente me compre un producto adicional? En ese caso, el modelo de clasificación nos podrá sugerir qué hacer para lograr el objetivo.

1.2 Comentarios Finales

Los modelos de clasificación no solamente responden a preguntas que impliquen que los individuos u objetos sean clasificados en dos categorías, las categorías pueden ser relativamente muchas. De hecho, los problemas de clasificación pueden ser menos tradicionales, como clasificar una imagen. Por ejemplo, para que un carro autónomo pueda funcionar, tiene que clasificar (entender) las señales de tránsito que "observa".

Si ya te es familiar los modelos de regresión múltiple[8], te parecerá que los modelos de clasificación se parecen en parte, pues se trata de emplear diferentes variables independientes (características de los individuos) para explicar una variable objetivo. Pero se diferencia de los modelos de regresión en que la variable dependiente de estos modelos es una variable cualitativa que toma relativamente pocos valores (Por ejemplo, `compra` o `no compra` o `pagará` o `no pagará`), mientras que la variable dependiente de los modelos de regresión corresponden a una variable cuantitativa continua o discreta.

Los modelos de clasificación que se emplean en el *business analitycs* tienen dos orígenes: modelos estadísticos y modelos de aprendizaje de máquina[9]. En este libro nos concentraremos en diferentes modelos que hacen parte de los dos campos mencionados anteriormente.

En el los Capítulos 3 y 5 estudiaremos los modelos *Logit* y clasificador bayesiano ingenuo (*Naive Bayes*) que tienen un origen estadístico. Por otro lado, en los Capítulos 6 a 9 estudiaremos modelos *machine learning* como el modelo **kNN**, Árboles de decisión,

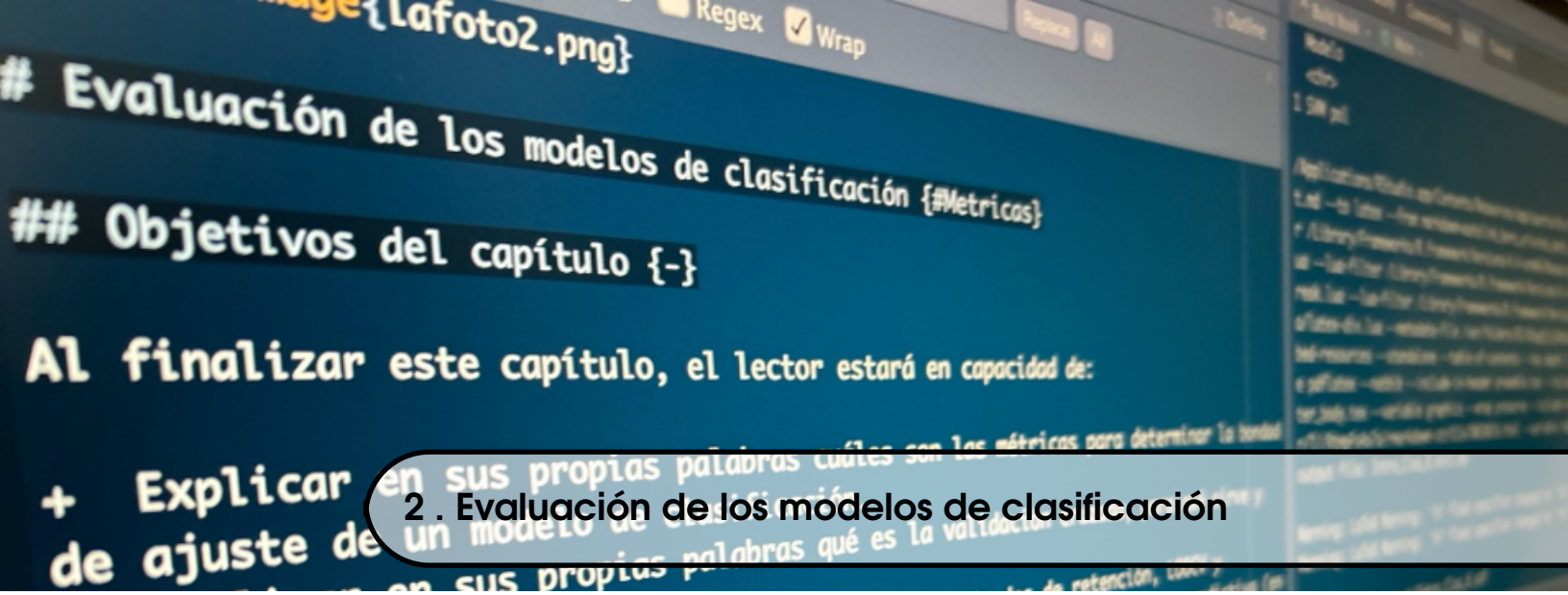
[7] Por una variable accionable en la jerga de los negocios, es aquella que le permite a la organización desarrollar estrategias y campañas que permitan el logro de un objetivo (Alonso, 2024).

[8] Ver Alonso (2024) para una discusión amplia del modelo clásico de regresión.

[9] También conocido como aprendizaje automático o aprendizaje automatizado. Este término viene del inglés *machine learning*.

el modelo *Random Forest* y *Suport Vector Machine*. Para cada uno de estos modelos, se aborda la parte formal del modelo junto con una aplicación en R (R Core Team, 2023).

Antes de iniciar a estudiar los diferentes modelos de clasificación que estudiaremos, es importante concentrar la atención sobre cómo determinar si un modelo es un buen modelo de clasificación. Eso lo estudiaremos en el Capítulo 2.



2 . Evaluación de los modelos de clasificación

Objetivos del capítulo

Al finalizar este capítulo, el lector estará en capacidad de:

- Explicar en sus propias palabras cuáles son las métricas para determinar la bondad de ajuste de un modelo de clasificación.
- Explicar en sus propias palabras qué es la validación cruzada, para qué sirve y cuándo se debe emplear.
- Efectuar en R una validación cruzada por los métodos de retención, LOOCV y *k-folds* para seleccionar un modelo de regresión para hacer analítica predictiva (en muestras de corte transversal).

2.1 Introducción

Siempre que se estima o entrena un modelo[1] surge la pregunta ¿qué tan bueno es el modelo?.

Cuando estamos trabajando con modelos cuya variable a explicar (variable objetivo) es cuantitativa (continua o discreta), la forma tradicional de medir qué tan bueno es un modelo es emplear la distancia entre el valor realmente observado y el valor "predicho" por el modelo; i.e., el error del modelo. En este ambiente es común emplear métricas como el **RMSE** o el R^2, como se realiza en la regresión múltiple o los modelos de series de tiempo.

Pero, cuando la variable a explicar (variable dependiente) es una variable categórica no tiene mucho sentido calcular un error y, por tanto, métricas como el **RMSE** o el R^2. Por ejemplo, supongamos que estamos hablando de un modelo que se estima o entrena para responder la pregunta ¿cuáles de los clientes pagarán la factura de la tarjeta de crédito a tiempo? En este caso la variable a explicar solo toma dos valores: `paga` y `no paga`. Si el modelo estimado o entrenado predice que un individuo pagará y en efecto esa persona no paga, en este caso tendremos un error, pero es imposible determinar la distancia entre el valor predicho y el valor realmente observado. Así, necesitaremos otro tipo de herramientas para determinar si un modelo es bueno o no. En este Capítulo nos concentraremos en algunas métricas de evaluación bastantes usadas en este campo y en discutir cómo se emplean para realizar validación cruzada de los modelos.

2.2 Estrategias para la validación cruzada de modelos

(Esta sección es una versión ligeramente modificada del capítulo 12 de Alonso (2024)).

Antes de entrar en la discusión de las métricas que se emplearán para decidir si un modelo es mejor que otro, es importante reconocer que independientemente de las métricas que se empleen para medir la bondad de ajuste de un modelo, éstas nos llevan a encontrar modelos que explican lo mejor posible la muestra bajo estudio, pero no necesariamente nuevas muestras que aparezcan. Es decir, es posible que tengamos un modelo muy bueno para explicar la muestra (analítica diagnóstica) pero no necesariamente para hacer analítica predictiva. Esto se conoce como el problema de *overfitting* (sobreajuste).

Cuando los científicos de datos están interesados en responder una pregunta de negocio que involucra emplear analítica predictiva, será necesario evaluar el poder predictivo del modelo. Es importante reconocer que la tarea de clasificación típicamente implica querer predecir la categoría de un individuo (analítica predictiva). Cuando queremos un buen modelo para predecir, queremos evitar el *overfitting* porque podemos estar dando demasiado poder predictivo a peculiaridades específicas de la

[1] En el mundo de la estadística se emplea la expresión "estimar un modelo" para la construcción de un modelo a partir de una muestra. Por otro lado, en el mundo de la inteligencia artificial se emplea la expresión "entrenar un modelo".

muestra que se empleó para estimar el modelo. Pero al mismo tiempo, queremos evitar tener un modelo con bajo ajuste a la muestra (infraajuste o *underfitting* en inglés) porque estaríamos ignorando patrones en la muestra útiles para determinar el comportamiento de un nuevo individuo.

Para evitar *overfitting* (sobreajuste) una buena práctica es emplear muestras diferentes para estimar y evaluar la capacidad predictiva de este. En general, cualquier técnica de validación cruzada de modelos implicará dividir la muestra en una muestra de evaluación que sea diferente a la muestra de estimación o también conocida como la muestra de entrenamiento. A esta práctica se le conoce como **validación cruzada** o en inglés *cross-validation*[2].

En general, para realizar la valoración del poder predictivo de varios modelos candidatos empleando la muestra disponible será necesario contar con:

1. una muestra de evaluación que sea diferente a la muestra de estimación o también conocida como la muestra de entrenamiento y
2. una métrica que permita determinar qué tan "precisas" son las predicciones de los modelos.

A continuación, discutiremos tres estrategias de validación cruzada comúnmente empleadas en la práctica por los científicos de datos.

2.2.1 Método de retención

La técnica de validación cruzada más sencilla implica seleccionar aleatoriamente unas observaciones de la muestra para hacer la estimación (muestra de estimación o muestra de entrenamiento) y otra para evaluar el comportamiento del modelo para predecir (muestra de evaluación). Esta técnica es conocida como el **método de retención** o *holdout method*.

En este caso, una parte relativamente pequeña de la muestra es seleccionada al azar[3] como muestra de evaluación y la muestra restante es la de estimación. Es común que se emplee el 80% de la muestra para la estimación y el 20% restante para realizar la evaluación. En la Figura 2.1 se presenta un diagrama de esta aproximación. Con la muestra de evaluación se comparan los diferentes modelos de regresión candidatos a ser el mejor modelo para predecir las observaciones. La comparación implica calcular diferentes métricas que discutiremos en la siguiente sección de este capítulo.

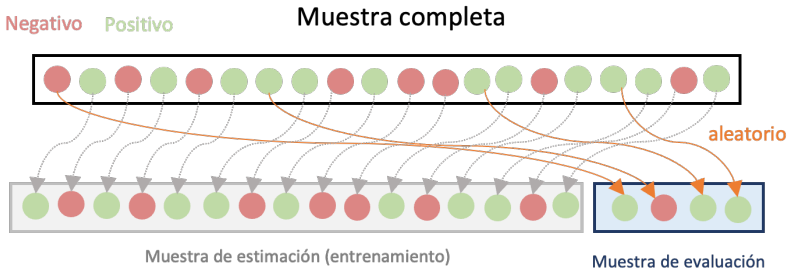
Una desventaja de esta aproximación es que la selección del mejor modelo para predecir podría estar determinada por el azar, pues la muestra seleccionada para la evaluación es totalmente aleatoria. Así, si se replicara el ejercicio con otra muestra, el mejor modelo seleccionado podría ser diferente.

Antes de continuar con otras aproximaciones de validación cruzada de modelos, definamos unos términos importantes en esta literatura. El error del modelo en la mues-

[2] Noten que esta técnica solo tiene sentido en datos de corte transversal, donde el orden de los datos no es importante.

[3] Por muestreo aleatorio sin reposición.

Figura 2.1. Diagrama del Método de retención para la evaluación cruzada de modelos



Fuente: elaboración propia.

tra de estimación se conoce como el **error de entrenamiento**. El error del modelo en la muestra de evaluación se conoce como el **error de prueba**.

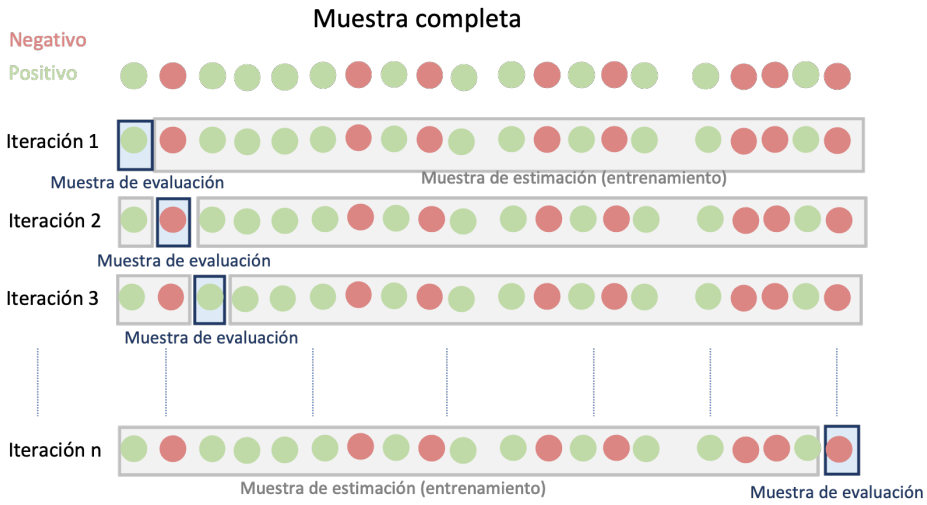
2.2.2 Método LOOCV

Otro método de valoración cruzada comúnmente empleado es el denominado LOOCV (por la sigla en inglés del término *Leave one out Cross-validation*). En este método, a diferencia del método de retención, no se emplea una única muestra de evaluación elegida al azar. Este método implica realizar la estimación y la valoración del modelo en diferentes muestras.

En este caso, se fija una observación en la muestra de evaluación y se estima el modelo con el resto de observaciones ($n - 1$). Este proceso se repite hasta que todas las observaciones de la muestra han sido empleadas en una muestra de evaluación. En la Figura 2.2 se presenta un diagrama de esta aproximación. Los modelos candidatos se comparan empleando el promedio de las métricas deseadas para evaluar las predicciones para cada una de las n muestras de evaluación.

A priori el método LOOCV parecería ser el ideal al emplear todas las observaciones como muestra de evaluación. Esto claramente eliminaría el problema de la aleatoriedad que genera el método de retención al momento de seleccionar el mejor modelo para predecir. No obstante, este método puede ser costoso computacionalmente pues requiere estimar el modelo n veces.

Figura 2.2. Diagrama del Método de validación cruzada de LOOCV para la evaluación de modelos



Fuente: elaboración propia.

2.2.3 Método de k iteraciones

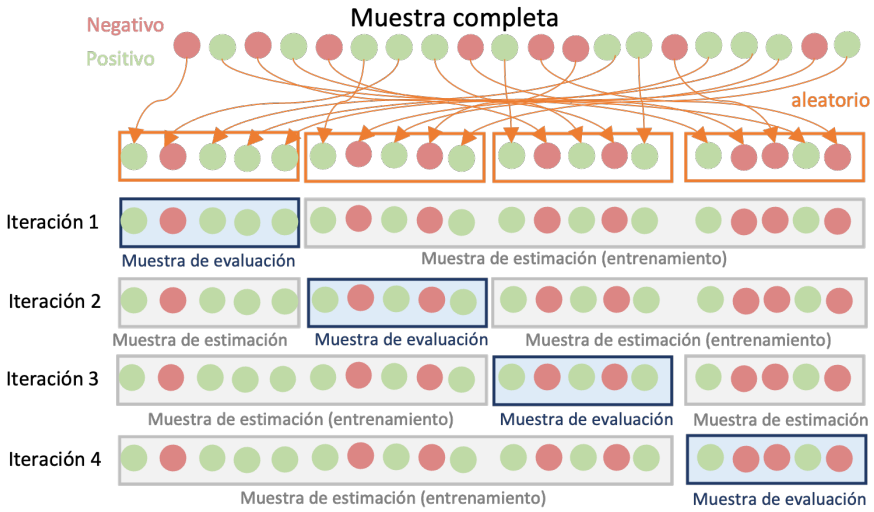
A parte del costo computacional del método LOOCV, puede existir potencialmente otro problema con esa aproximación. El LOOCV puede generar una mayor variación en el error de predicción si algunas observaciones son atípicas. Para evitar este problema, lo ideal sería utilizar una buena proporción de observaciones como muestra de prueba para evitar el peso de las observaciones atípicas.

El método de **validación cruzada de k iteraciones** o *k-fold Cross-validation* intenta ser un intermedio entre el método de retención y el LOOCV. Este método implica dividir de manera aleatoria la muestra completa en $kgrupos$ de aproximadamente el mismo tamaño. Para cada uno de los k grupos (o iteraciones) se emplean los restantes $k-1$ grupos como muestra de estimación y el grupo k de observaciones se emplea como muestra de evaluación para la cual se calculan las respectivas métricas deseadas para los modelos a comparar. Y finalmente, para obtener la métrica para todo el ejercicio se calcula el promedio de las k métricas calculadas para cada modelo. En la Figura 2.3 se presenta un diagrama de esta aproximación. Lo más común en la práctica es emplear un valor de k de 5 o 10.

En la práctica, *la validación cruzada de k iteraciones se recomienda generalmente sobre los otros dos métodos* debido a su equilibrio entre la variabilidad que puede aparecer por los datos atípicos (método LOOCV), el sesgo fruto de emplear sólo una muestra de evaluación (método de retención) y el tiempo de ejecución computacional. En el Capítulo 10 se presenta una aplicación de este método de validación cruzada.

En la siguiente sección discutiremos las métricas para valorar la precisión de las pre-

Figura 2.3. Diagrama del Método de validación cruzada de k iteraciones para la evaluación de modelos



Fuente: elaboración propia.

dicciones.

2.3 Medidas de ajuste de los modelos de clasificación

Evaluar el desempeño de un modelo de clasificación no es una tarea trivial. En este caso, construir un error como la diferencia entre el valor que el modelo espera y el que realmente se observa no es viable, como si lo es en la tarea de regresión o de proyectar.

En los modelos de regresión y pronósticos típicamente se guarda una parte de la muestra (muestra de evaluación) para determinar si el modelo que ha sido entrenado (estimado) en la muestra de entrenamiento (de estimación) tiene un buen comportamiento en ésta. Es decir, se emplea una muestra de entrenamiento para estimar el modelo. Y posteriormente los modelos candidatos son comparados en la muestra de evaluación con una medida de qué tan "cerca" están las predicciones del modelo en la muestra de evaluación. Es decir, se emplean medidas de la distancia (error) entre el valor observado y el valor esperado por el modelo. Entre más pequeña se la medida del error que se emplea, mejor será el modelo. Esto siempre será posible con este tipo de modelos que implican un aprendizaje supervisado sobre variables cuantitativas.

Lastimosamente, en la tarea de clasificación esto no es posible. Dado que las observaciones de la variable objetivo solo corresponden a dos (o un número reducido) de categorías. Así, en este caso calcular la distancia entre la predicción de la categoría y la categoría observada no tiene sentido. Pues en el mundo de las variables cualitativas

(que representan las categorías) no tiene sentido restar dos números.

Por eso, es necesario tener una aproximación diferente para determinar qué modelo es mejor para una determinada muestra, independientemente de la estrategia de validación cruzada que empleemos. Los modelos de clasificación producen dos tipos de predicciones. Unos modelos producen como predicción la probabilidad de que ocurra una clase (por ejemplo, el modelo *Logit* que estudiaremos en el Capítulo 3. Para estos modelos se requiere un paso intermedio para pasar de la probabilidad de que ocurra una determinada categoría a la predicción de la categoría. En estos casos, se establece un valor de corte o umbral (en inglés *Cutoff Value* o *Treshold Value*) para determinar a partir de qué valor de la probabilidad estimada se asignará la predicción a una categoría. Para este tipo de modelos se deben tener unas consideraciones especiales que discutiremos en la Sección 2.3.1. Posteriormente discutiremos las métricas que se pueden emplear una vez se cuenta con la predicción de la categoría de los individuos.

2.3.1 Consideraciones para los modelos de clasificación que generan probabilidades

Como se discutirá a lo largo de este libro, algunos de los modelos o algoritmos de clasificación no generan directamente una predicción de la categoría de un individuo, sino la probabilidad (condicionada a las características de este) de que un individuo pertenezca a una categoría determinada. Para este tipo de modelos tenemos que tener en cuenta unas consideraciones especiales a diferencia de los algoritmos que generan directamente las predicciones de la categoría.

De esta manera para los modelos que generan una probabilidad de pertenecer a una categoría, tendremos que construir una forma para decidir a partir de qué probabilidad clasificaremos a un individuo como perteneciente a una categoría o no.

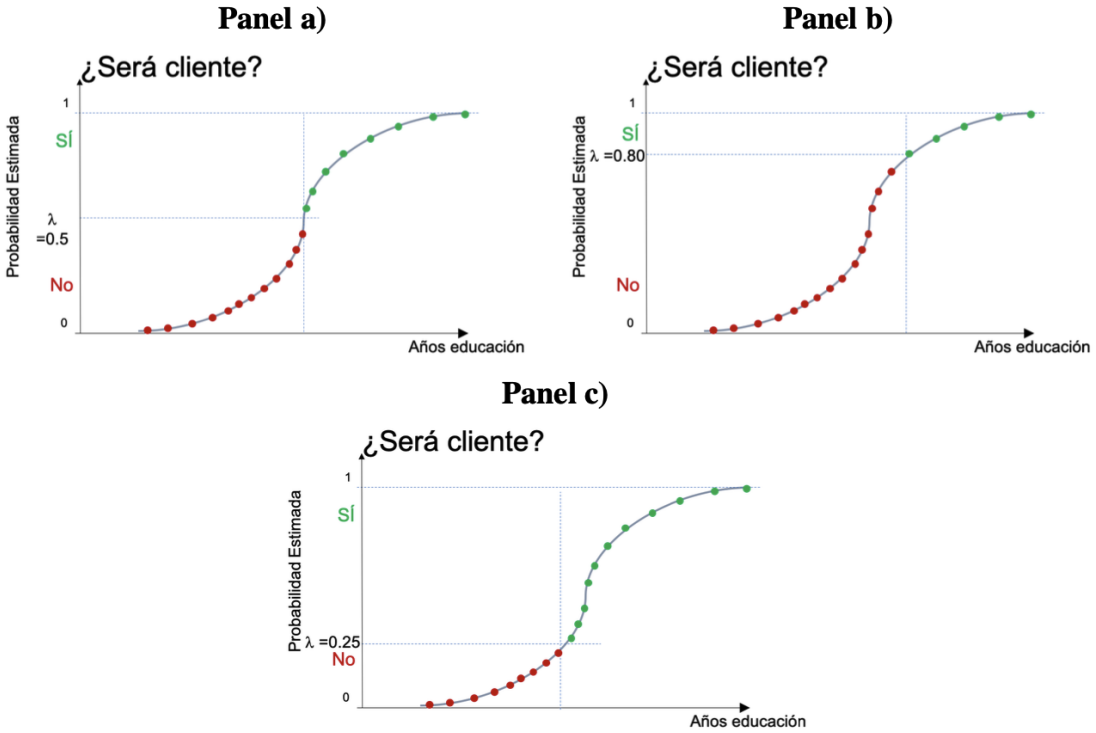
Partamos del modelo más sencillo, donde queremos clasificar solo en 2 categorías, positivos y negativos[4]. Es decir, nuestra variable objetivo (y_i) puede tomar dos valores: positivo (y_i=1) o negativo (y_i=0). Los diferentes modelos nos darán la probabilidad de que observemos un valor positivo ($y_i = 1$). Entonces, para determinar el valor estimado del modelo para y_i ($\hat{y}_i$) tendremos que definir un valor de corte[5] λ para determinar a partir de qué valor de la probabilidad estimada se entenderá que $\hat{y}_i$ tomará el valor de uno.

Es común que la primera aproximación sea definir a $\lambda = 0{,}5$, pero esta selección es arbitraría. λ puede ser seleccionada por el científico de datos para ajustar la precisión del modelo. En la Figura 2.4 se presentan diferentes valores de corte (λ). Noten que los valores predichos como uno o cero para un individuo puede cambiar dependiendo del valor de corte seleccionado. Esto nos lleva a la necesidad de evaluar para un modelo, diferentes λ, pues la aproximación se puede comportar de manera diferente para cada punto de corte para la probabilidad. En la siguiente sección se discuten las medidas de ajuste que emplearemos una vez se tengan los valores estimados para y_i.

[4]Es importante resaltar que, en esta literatura cuando la variable dependiente toma el valor de uno se le conoce como un resultado positivo y en caso de tomar un cero se le denomina negativo. De aquí en adelante adoptaremos ese lenguaje por simplicidad.

[5]El Valor de corte también es conocido como el umbral, *Cutoff Value* o *treshold Value*.

Figura 2.4. Ejemplo de diferentes valores de λ



Fuente: elaboración propia.

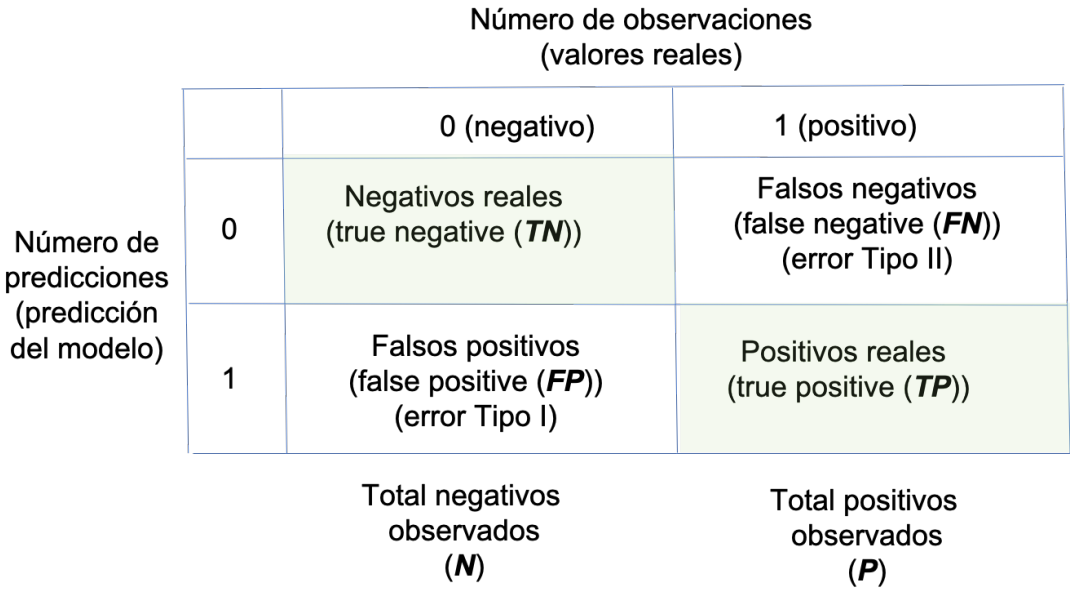
2.3.2 Medidas de ajuste del modelo basadas en la "exactitud" de la predicción

Una vez contamos con la clasificación construida por cada modelo para los individuos en unos $\hat{y}_i = 1$ y ceros $\hat{y}_i = 0$, podemos emplear diferentes métricas que intentan constatar que tan cerca está el modelo para predecir de manera exacta la clasificación del individuo. Para esto, se comparará la predicción con lo realmente observado en la muestra de evaluación.

2.3.2.1 Matriz de confusión

Para un determinado modelo[6] podemos construir una matriz que en cada fila reporte el número de predicciones de cada clase, y en cada columna se registre los valores observados (reales) en la muestra de evaluación para cada clase. A este tipo de matriz se le conoce como **matriz de confusión**. En la Figura 2.5 se presenta un ejemplo de matriz de confusión para un caso en el que solo existen dos posibles clases.

Figura 2.5. Ejemplo matriz de confusión



Fuente: elaboración propia.

Ahora definamos las siguientes cantidades:

- N: Total de negativos observados.
- P: Total positivos observados.
- TN: El número de negativos reales. Es decir, el número de predicciones negativas (0) que son iguales al valor real observado (true negative).
- TP: El número de positivos reales (true positive).

[6] Los modelos o algoritmos pueden o no implicar un correspondiente valor de corte λ.

- FN: El número de falsos negativos (false negative). Es decir, el número de predicciones negativas (0) que no aciertan. Esto se conoce en estadística como el error tipo II.
- FP: El número de falsos positivos (false positive). Esto se conoce en estadística como el error tipo I.

La matriz de confusión puede ser muy útil si nos concentramos en la diagonal principal (TN y TP). Pero en algunos casos puede ser muy difícil comparar dos matrices de confusión de modelos candidatos, en especial cuando empezamos a tener diferentes valores entre los modelos en las posiciones por fuera de la diagonal principal. En otras palabras, al comparar matrices de confusión tendremos que monitorear lo que ocurre en las cuatro celdas de la matriz. Sería más fácil contar con un solo número que sintetice lo que ocurre en las cuatro celdas de la matriz. Por eso se han desarrollado diferentes medidas que permiten comparar modelos de clasificación. Estas métricas se presentan a continuación.

2.3.2.2 Sensibilidad

La **sensibilidad** representa la capacidad del modelo de clasificación de detectar correctamente los valores verdaderos. La sensibilidad también es conocida como **recall**, **tasa de acierto** (*hit rate*) o **tasa de verdaderos positivos** (*true positive rate* TPR).

Consideremos un ejemplo en el que se quieren clasificar unos individuos dadas sus características en dos categorías: `los clientes` y `los no clientes`. En este caso la sensibilidad del modelo sería la capacidad del algoritmo de identificar correctamente los clientes.

La sensibilidad se define como:

$$TPR = \frac{TP}{P} \tag{2.1}$$

En otras palabras, la sensibilidad representa la fracción de verdaderos positivos. Es decir, la probabilidad (no condicionada) de que el modelo de clasificación identifique correctamente los unos.

2.3.2.3 Especificidad

Por otro lado, la **especificidad** (TNR por la sigla en inglés del término tasa de verdaderos negativos) se centra en la otra esquina de la matriz de confusión. Esta corresponde a la probabilidad de que un individuo al que se le observa un cero sea clasificado como un cero. Es decir, cuál es la proporción de `no clientes` que han sido correctamente clasificados por el modelo como `no cliente`. La especificidad se define como:

$$TNR = \frac{TN}{N} \tag{2.2}$$

La especificidad también se conoce como **selectividad** (*selectivity*).

2.3.2.4 Precisión

La **precisión** o **valor positivo predicho** (PPV por su sigla en inglés) corresponde a la proporción de verdaderos positivos de todos los positivos que predice el modelo. Es decir,

$$PPV = \frac{TP}{TP + FP} \tag{2.3}$$

En otras palabras, el PPV corresponde a la probabilidad de que un positivo predicho por el modelo sea un verdadero positivo. Para nuestro ejemplo de los clientes, el PPV corresponde a la probabilidad de que uno de los clientes que es predicho por el modelo sea efectivamente un cliente.

2.3.2.5 Valor negativo predictivo

El **valor negativo predictivo** (NPV por su sigla en inglés) corresponde a la proporción de verdaderos negativos de todos los negativos que predice el modelo. Es decir,

$$NPV = \frac{TN}{TN + FN} \tag{2.4}$$

En nuestro ejemplo, el NPV es la probabilidad de que un no cliente predicho por el modelo sea realmente un no cliente.

2.3.2.6 Exactitud

La **exactitud** (ACC) (o conocido por su nombre en inglés *accuracy*)[7] se utiliza también como una medida de qué tan bien se comporta un modelo de clasificación. Esta medida tiene en cuenta tanto elementos que se miden en la sensibilidad (*recall*) y en la especificidad. Se concentra en la proporción de aciertos tanto en negativos como en positivos. Es decir, en la diagonal principal de la matriz de confusión. La exactitud se mide como:

$$ACC = \frac{TP + TN}{P + N} \tag{2.5}$$

En otras palabras, la exactitud es la proporción de resultados acertados (tanto verdaderos positivos (TP) como verdaderos negativos (TN)) entre el número total de observaciones.

2.3.2.7 Puntaje F_1

Otra métrica de la bondad del modelo que emplea directamente la sensibilidad (*recall*) y la especificidad es el puntaje F_1 (F_1 score). Este indicador se calcula de la siguiente manera:

$$F_1 = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{2.6}$$

[7] Esta medida también es conocida en la literatura como **valor predictivo positivo**, **exactitud Rand** o **índice Rand**.

Esta métrica corresponde a un promedio armónico (no aritmético) de la sensibilidad (*recall*) y la especificidad. La intuición detrás de este indicador es que al mismo tiempo que se está buscando un modelo que tenga una proporción alta de negativos correctamente predichos (alta precisión) se busca un porcentaje de todos los positivos cubiertos por el modelo (alta sensibilidad).

Recordemos que un indicador alto de precisión implica una mayor capacidad del modelo para clasificar los positivos correctamente. La combinación de esto con el *recall* da una idea de cuántos del total de positivos puede cubrir el modelo.

2.3.2.8 ROC y AUC para modelos con umbrales λ

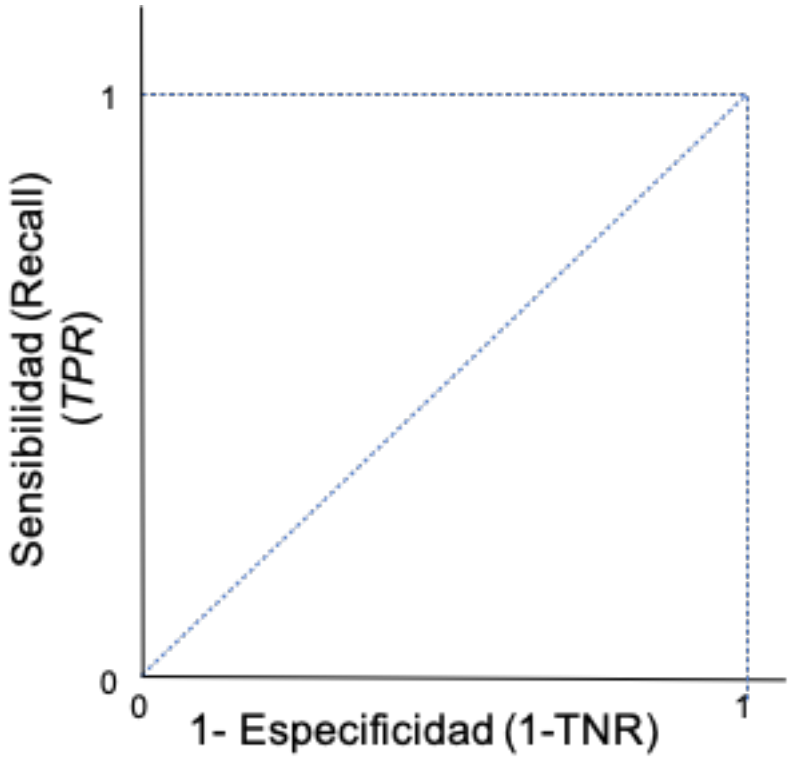
La sensibilidad (TPR) y la especificidad (TNR) son inversamente proporcionales entre sí. Entonces, cuando aumentamos la sensibilidad, la especificidad disminuye y viceversa. Para modelos que proveen una probabilidad de positivos y no directamente una predicción de un positivo o negativo, se tiene que, cuando disminuimos el umbral λ, obtenemos más valores positivos predichos (ver Figura 2.4 panel b), por lo que aumenta la sensibilidad y disminuye la especificidad. Del mismo modo, cuando aumentamos el umbral, obtenemos más valores negativos (ver Figura 2.4 panel c) y, por lo tanto, obtenemos una mayor especificidad y una menor sensibilidad. Las otras medidas discutidas anteriormente también se verán afectadas por el cambio de punto de corte, pero el impacto no es tan claro.

De la discusión anterior es claro que elegir el mejor modelo de clasificación implica encontrar un equilibrio entre predecir con precisión los positivos y los negativos. En otras palabras, un balance entre la sensibilidad (*recall*) y la especificidad. Así mismo, estas dos cantidades dependen del punto de corte.

La **curva de ROC** (por la sigla del término en inglés *Receiver Operating Characteristics*) tiene como intención recoger este resultado. Para graficar la curva **ROC** emplearemos en el eje horizontal la proporción de falsos negativos, que corresponde a uno menos la proporción de verdaderos negativos predichos por el modelo. En otras palabras, uno menos la especificidad. En el eje vertical se presenta la sensibilidad (*recall* o tasa de positivos reales) (Ver Figura 2.6). La curva **ROC** muestra la combinación de proporción de falsos negativos y sensibilidad para cada determinado valor de corte.

Ahora pensemos en la situación ideal. Una situación ideal es un modelo que permite generar una distribución de negativos (TN) y positivos (TP) que discrimine completamente los dos posibles estados como la que se describe en el panel a) de la Figura 2.7. Ahora supongamos un $\lambda = 0{,}5$ (punto de corte). En este caso, la proporción de ceros falsos negativos es cero y la proporción de tasa de positivos reales (sensibilidad) es uno. Esta situación la representa un punto verde en la Figura 2.7 panel b) que se encuentra en el punto (0,1). Ahora, supongamos que aumentamos el punto de corte a $\lambda = 0{,}75$. En este caso, la sensibilidad disminuirá; al mismo tiempo la proporción de ceros falsos negativos sigue siendo cero. Esto se representa con el punto verde sobre el eje vertical (ver Figura 2.7 panel b)). Ahora, supongamos que el punto de corte se disminuye a 0.25 ($\lambda = 0{,}25$). En ese caso la sensibilidad será 1 (proporción de verdaderas positivos predichos) y la proporción de falsos negativos aumentará. Esto se representa con un punto verde (ver Figura 2.7 panel b)). Así podemos seguir obteniendo diferentes pun-

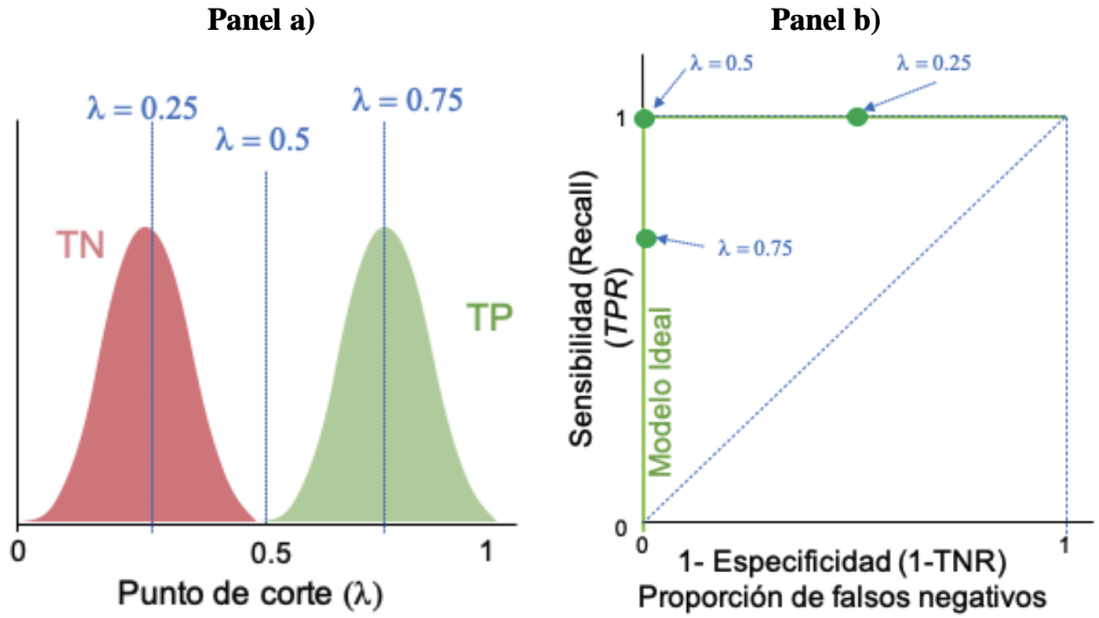
Figura 2.6. Ejes de la Curva ROC



Fuente: elaboración propia.

tos para diferentes puntos de corte. La unión de todos esos puntos es la curva **ROC**. En este caso el mejor de los escenarios implica una curva **ROC** que será una línea recta vertical de (0,0) hasta (0,1) y posteriormente vertical de (0,1) hasta (1,1).

Figura 2.7. Curva ROC del mejor modelo posible



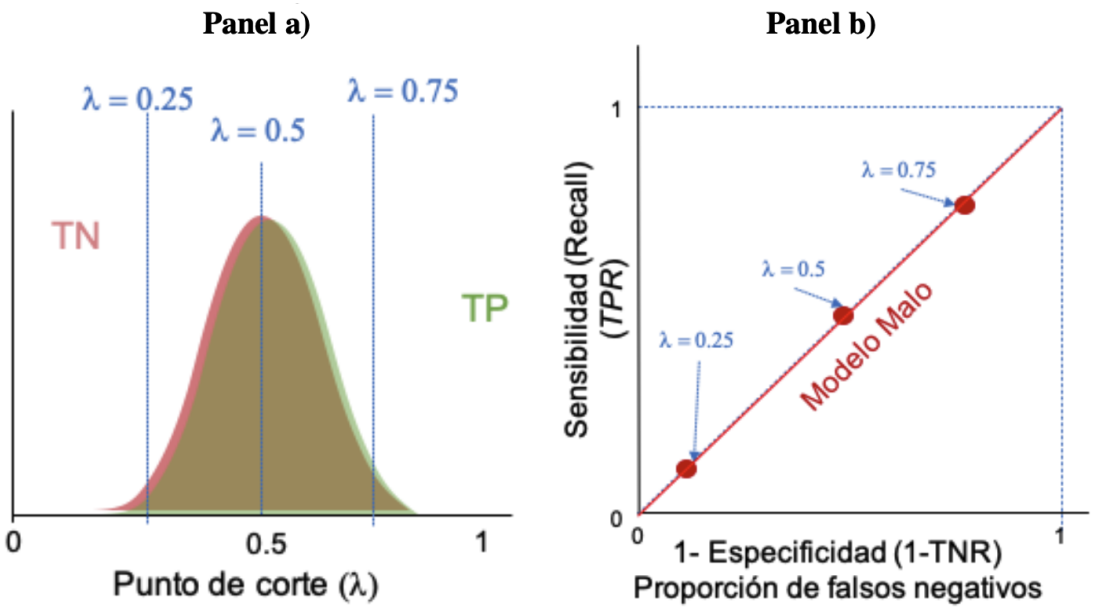
Fuente: elaboración propia.

Ahora, supongamos un escenario base en el que se tiene un modelo de clasificación totalmente aleatorio, en el que no se sabe nada de los datos a clasificar. Ese modelo clasificaría de manera aleatoria a cada individuo mandándolos a cada una de las dos categorías con la misma distribución de probabilidad, como la que se describe en el panel a) de la Figura 2.8. En ese caso, un punto de corte de 0.5 implicará una proporción de falsos negativos igual a la sensibilidad (TPR). Esto se puede observar con un punto rojo en la Figura 2.8 panel b). Si aumentamos (disminuimos) el punto de corte la proporción de falsos negativos y la sensibilidad aumentarán (disminuirán). Si se realiza el mismo ejercicio para todos los valores de corte posibles, obtendremos una curva **ROC** que va desde el origen (0,0) hasta el punto (1,1) (ver Figura 2.8 panel b)). Esta línea se considera la línea de base para comparar los modelos de clasificación.

En la práctica es poco común obtener curvas **ROC** como las descritas en las Figuras 2.7 y 2.8. Es más común observar curvas **ROC** como la descrita en la Figura 2.9.

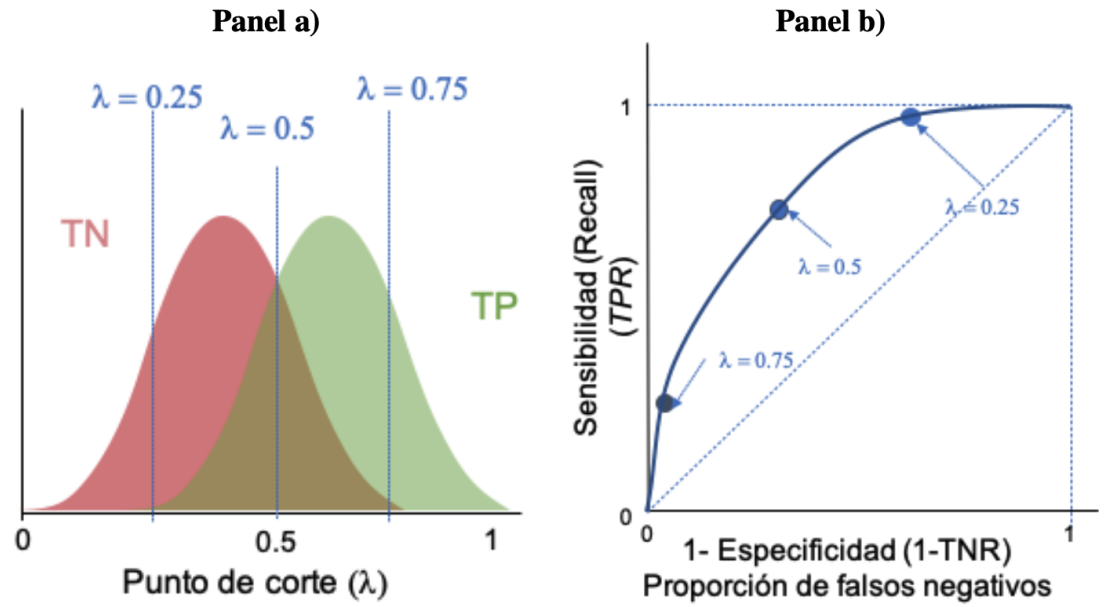
En algunos casos es complicado hacerle seguimiento a toda la curva para comparar varios modelos. Por ejemplo, las curvas **ROC** de dos modelos se pueden cruzar. Un indicador que resume la curva **ROC** es el **Área Bajo la Curva ROC** (**AUC** por la sigla en inglés *Area under curve*). Noten que en el caso ideal presentado en la Figura 2.7 panel b) el área bajo la curva es 1 (corresponde a un cuadrado con lados 1). En el caso del

Figura 2.8. Curva ROC de un modelo aleatorio base



Fuente: elaboración propia.

Figura 2.9. Curva ROC de un modelo típico de clasificación



Fuente: elaboración propia.

modelo de clasificación aleatorio que sirve como línea de base para un mal modelo (ver Figura 2.8 panel b), el **AUC** es $\frac{1}{2}$ (el área del triángulo con base 1 y altura 1). Para la curva **ROC** presentada en la Figura 2.9 el **AUC** estará entre 0.5 y 1. En general el AUC entre más grande mejor será el modelo. Es relativamente común emplear la siguiente regla empírica para decidir sobre la bondad de ajuste del modelo:

- $AUC = 0{,}5$: Es un mal clasificador, es como lanzar una moneda.
- $0{,}5 \leq AUC < 0{,}6$: Modelo de clasificación es malo.
- $0{,}6 \leq AUC < 0{,}75$: Modelo de clasificación es regular.
- $0{,}75 \leq AUC < 0{,}9$: Modelo de clasificación es bueno.
- $0{,}9 \leq AUC < 0{,}95$: Modelo de clasificación es muy bueno.
- $0{,}97 \leq AUC < 1$: Modelo de clasificación es excelente.
- $AUC = 1$: Modelo de clasificación es perfecto.

Noten que otra cosa importante que nos muestra la curva **ROC** para un modelo que genera una predicción de la probabilidad de ser un positivo de un individuo es el punto de corte (λ) que da la mejor combinación de sensibilidad y especificidad. El punto de la **ROC** que se aleje mas de la linea de base o que se acerque más al punto (0,1) será un buen candidato de punto de corte. Si bien existen varios criterios para encontrar el punto de corte óptimo, un método conocido es maximizar el **Índice de Youden** (Youden, 1950). Este índice se define como la suma de la **selectividad** y **especificidad** menos uno. Es decir:

$$J = TPR + TNR - 1 \tag{2.7}$$

Así, encontrar el punto de corte óptimo implicará calcular J para todos los posibles λ y seleccionar el λ donde el J se maximice.

2.4 Comentarios finales

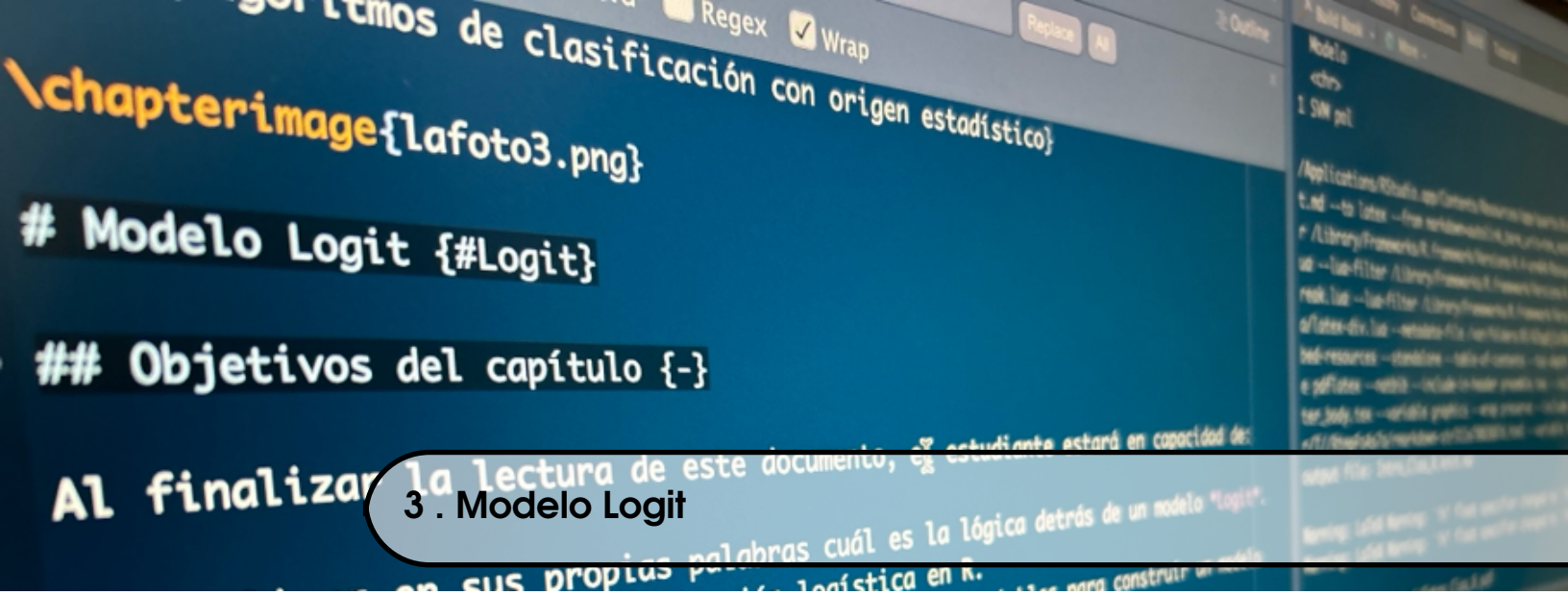
La selección de la o las métricas adecuadas para la evaluación de los modelos dependerá de la pregunta de negocio y de lo que es deseable para cada organización.

Por otro lado, como se discutió anteriormente, las métricas se emplearán típicamente sobre muestras de evaluación y no de estimación o entrenamiento, para evitar el *overfitting* (sobreajuste). Así, como la métrica adecuada, se deberá escoger el esquema de **validación cruzada** que se empleará.

En los siguientes capítulos estudiaremos diferentes modelos de clasificación y emplearemos las técnicas de evaluación (métricas y estrategia de validación cruzada) para los modelos de clasificación estudiadas en este Capítulo.

Parte II

Algoritmos de clasificación con origen estadístico



3 . Modelo Logit

Objetivos del capítulo

Al finalizar la lectura de este documento, el estudiante estará en capacidad de:

- Explicar en sus propias palabras cuál es la lógica detrás de un modelo *Logit*.
- Estimar un modelo de regresión logística en R.
- Emplear técnicas de selección automática de variables para construir un modelo *Logit*.

3.1 Generalidades

El modelo *Logit* es una generalización del modelo de regresión lineal múltiple[1]. El modelo de regresión lineal implica una variable dependiente que es continua (o que se supone continua)[2], en el caso de los modelos de clasificación, éstos tendrán una variable cualitativa como variable dependiente. Por ejemplo, si nos concentramos en los casos de clasificación en los que solo hay dos categorías, la variable dependiente será una variable dummy[3] (dicotómica). Ejemplos de estas variables son: i) si un individuo pertenecerá o no a un grupo, ii) si un individuo será cliente o no, iii) si un individuo pagará o no.

Una posibilidad para estimar este tipo de modelos es emplear el modelo lineal y estimarlo por MCO. Esta aproximación de emplear el modelo lineal con una variable dependiente se conoce como el Modelo de Probabilidad Lineal (MPL). Lastimosamente esta aproximación de estimar por MCO el MPL tiene varios problemas, unos solucionables y otros no.

Los problemas más comunes de estimar modelos con variables dummy como dependientes por MCO son:

1. El error del modelo presenta problemas de heteroscedasticidad[4]. De hecho se sabe que en este caso: $\sigma_i^2 = E\left[\mathbf{y_i}|\mathbf{x}_i\right]\left(1 - E\left[\mathbf{y_i}|\mathbf{x}_i\right]\right)$, donde $\mathbf{x}_i$ representa un vector fila de dimensiones $(1 \times k)$ que recoge las características del individuo i[5].
2. El modelo estimado no garantiza que los valores estimados para la variable dependiente estén en el intervalo [0,1]. Esto implica, por ejemplo, varianzas negativas.

El primero de los problemas se puede solucionar empleando el método de Mínimos Cuadrados Ponderados (MCP).[6] El segundo problema no tiene una solución fácil y hace que este modelo no sea muy usado en la actualidad. Para entender en detalle este problema empleemos un ejemplo.

Supongamos que se desea explicar el hecho de que un individuo será cliente o no por medio de sólo una variable: los años de educación ($X_{2,i}$), Es decir, se tiene que $\mathbf{x}_i = [1, X_{2i}]$, por tanto:

$$y_i = \beta^T \mathbf{x}_i^T + \varepsilon_i y_i = \beta_1 + \beta_2 X_{2,i} + \varepsilon_i \tag{3.1}$$

[1] Para una descripción detallada del modelo de regresión clásico ver Alonso (2024).

[2] Como por ejemplo las unidades demandadas de carros; que si bien no es una variable continua, se supone como tal.

[3] Para una introducción a las variables dummy y su uso, consultar el Capítulo 5 de Alonso (2024).

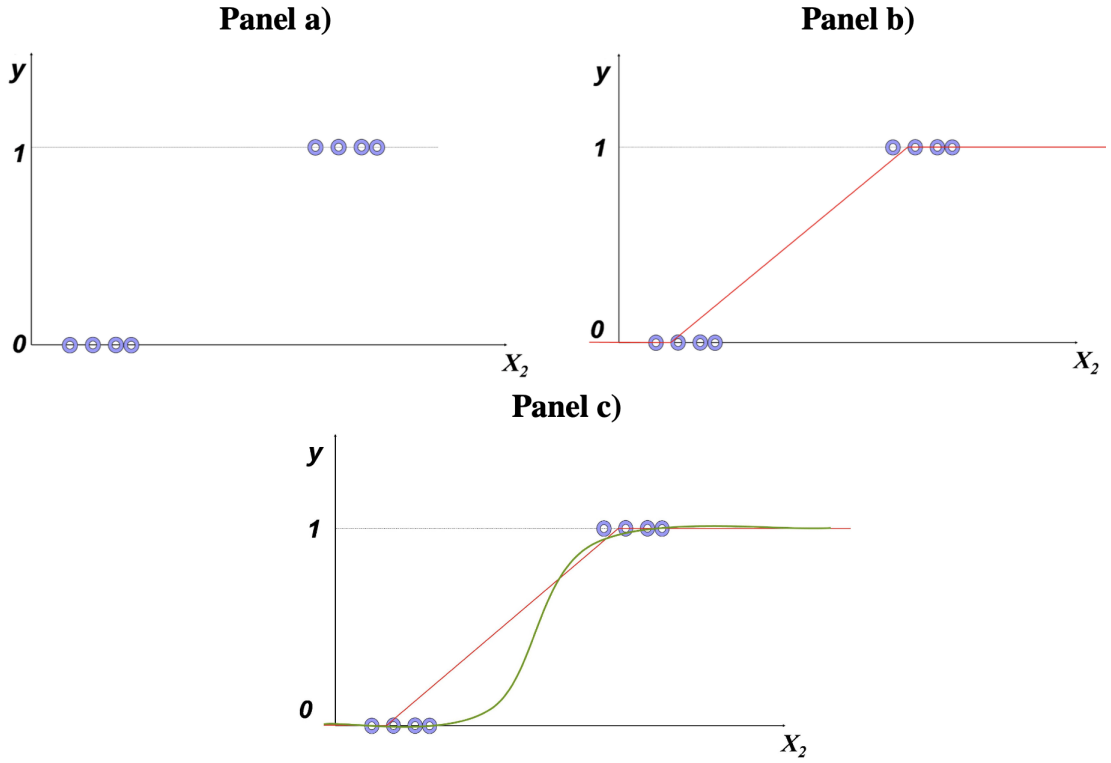
[4] En el apéndice 3.5.1 se presenta una prueba de esta afirmación.

[5] Es decir, corresponde a la fila i de la matriz $\mathbf{x}_i$.

[6] Es importante anotar que para emplear el método de MCP será necesario emplear como variable ponderadora $\omega_i^2 = E\left[\mathbf{y_i}\left|\mathbf{x}_i\right.\right]\left(1 - E\left[\mathbf{y_i}\left|\mathbf{x}_i\right.\right]\right)$. Naturalmente, en el caso del MPL se tiene que $E\left[y_i\left|\mathbf{x}_i\right.\right] = \beta^T \mathbf{x}_i^T$. Dado que el vector β no es conocido, es imposible calcular ω_i. Esto hace necesario estimar esta variable ponderadora en un primer paso. Así, emplear el método de MCP implicará un primer paso para estimar el modelo original por MCO para obtener $\hat{\beta}_{OLS}$. Con estos coeficientes estimados se puede encontrar la variable ponderadora $W_i = \sqrt{\hat{y}_i\left(1 - \hat{y}_i\right)}$, donde $\hat{y}_i = \left(\hat{\beta}_{OLS}\right)^T \mathbf{x}_i^T$. El segundo paso será transformar los datos dividiéndolos por W_i. El tercer paso implica estimar el modelo por MCO con las variables transformadas.

Así, para estimar el modelo se tendrá que recoger datos de individuos que son clientes ($y_i = 1$) y aquellos que no ($y_i = 0$) y los respectivos años de educación de cada individuo. Al graficar estos datos tendremos algo parecido al panel a) de la Figura 3.1. Noten que sólo se tendrán observaciones de la variable dependiente con valores de 1 y 0. El método de MCO lo que intentará hacer es encontrar una línea recta que minimice la suma de la distancia cuadrada entre los puntos observados y la línea estimada. Así, se obtendrá una línea como la presentada en el panel b) de ese mismo gráfico. Noten que el modelo estimado ($\hat{y}_i = \hat{\beta}_1 + \hat{\beta}_2 X_{2,i}$) no acota los valores estimados para la variable dependiente al intervalo [0,1] .

Figura 3.1. Modelo lineal con variable dummy como dependiente



Fuente: elaboración propia.

Además, este gráfico permite entender que un modelo lineal no podrá explicar de manera adecuada este tipo de modelos en los que la variable dependiente solo toma dos valores. De hecho, parece más adecuado emplear modelos no lineales para ajustar los datos, tal como se muestra en el panel c) de la Figura 3.1. Un modelo no lineal que se pueden emplear en estos casos es el modelo *Logit*[7].

Una manera de asegurar que los valores estimados del modelo se mantengan en el

[7]También podría emplearse el modelo *Probit* que es muy similar pero no es tan empleado por los científicos de datos. Los resultados del modelo *Probit* son muy similares a los del modelo *Logit*.

intervalo [0,1] es emplear una forma funcional acorde. Esto se garantiza empleando funciones de probabilidad acumulativa. Para entender cómo emplear estas funciones de probabilidad, empleemos el mismo ejemplo anterior, pero permitamos la posibilidad de más variables explicativas.

Es decir, suponga que se quiere explicar la variable y_i, tal como se definió en nuestro ejemplo, empleando $k-1$ variables explicatorias y una constante. Así, emplearemos el vector fila $\mathbf{x}_i$ de dimensiones $(1 \times k)$, que recoge las características del individuo i para explicar la probabilidad de ser cliente.

Un individuo que está decidiendo ser cliente o no, tomará su decisión dependiendo de si el beneficio neto de ser cliente es positivo o no. Así, si el beneficio neto de ser cliente es positivo, el individuo participará y si éste es negativo, entonces el individuo no participará. Por ahora definamos el beneficio neto del individuo i como Z_i. Noten que esta variable no es observable más adelante discutiremos esto. Así, tendremos que:

$$y_i = \begin{cases} 1 & \text{si } Z_i > 0 \\ 0 & \text{si } Z_i \leqslant 0 \end{cases} \tag{3.2}$$

Pero, ¿de qué depende ese beneficio neto? Supongamos que este es función de las características del individuo recolectadas en el vector $\mathbf{x_i}$. Es decir:

$$\mathbf{Z}_i = \beta^T \mathbf{x}_i^T + \varepsilon_i$$

Donde ε_i es un término aleatorio de error. Así, para unas características dadas para el individuo i tendremos que el valor esperado del beneficio neto será:

$$E\left[Z_i \left| \mathbf{x}_i \right.\right] = \bar{Z}_i = \beta^T \mathbf{x_i^T}$$

Recapitulando, las características del individuo i ($\mathbf{x}_i$) afectan el beneficio neto (Z_i) y éste a su vez explica si el individuo es cliente o no; es decir, y_i. Ahora bien, noten que la distribución del beneficio neto, dadas unas características del individuo i ($\mathbf{x}_i$), dependerá de la distribución del término de error ε_i.

Por conveniencia, podemos emplear el hecho que $Z_i = \beta^T \mathbf{x}_i^T + \varepsilon_i > 0$ implica que $\varepsilon_i > -\beta^T \mathbf{x}_i^T = -Z_i$ para reescribir la ecuación (3.3) de la siguiente manera:

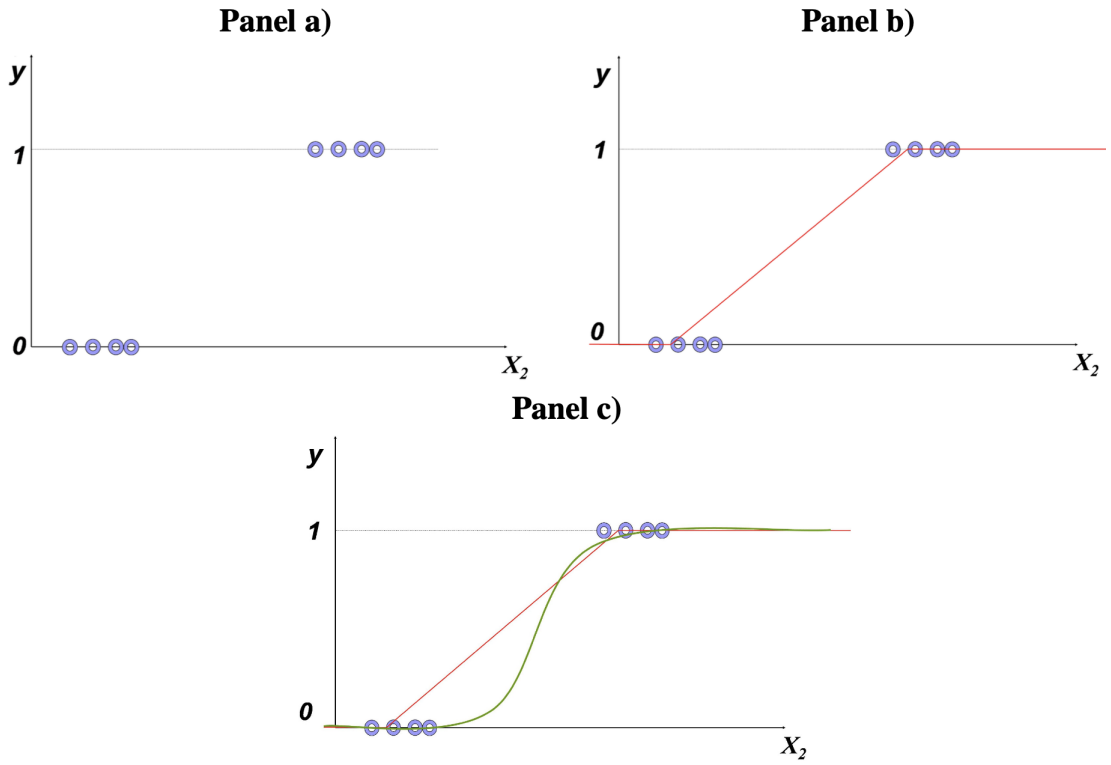
$$y_i = \begin{cases} 1 & \text{si} - Z_i < 0 \\ 0 & \text{si} - Z_i \geqslant 0 \end{cases} \tag{3.3}$$

En el panel a) de la Figura 3.2 se puede observar un esquema de un individuo con una relativa baja probabilidad de ser cliente, dadas sus características; mientras que en el panel b) se puede observar un individuo con una probabilidad relativamente alta. Es importante anotar que la probabilidad está expresada por el área bajo la curva de la función de distribución y dependerá de la forma de ésta. Finalmente, cabe anotar que la función que determina el área bajo la curva a la izquierda de cualquier

valor determinado de una función de distribución se conoce como la función de distribución acumulativa. Es decir,

$$F\left(Z_i\right) = P\left(Z_i \leq c\right) = \int_{-\infty}^{c} f\left(t\right) dt$$

Figura 3.2. Distribución del beneficio neto (de la variable latente)



Fuente: elaboración propia.

En otras palabras se tendrá que:

$$P\left[Y_i = 0|\mathbf{x}_i\right] = F\left(-Z_i\right) = F\left(-\beta^T \mathbf{x}_i^T\right)$$

$$P\left[Y_i = 1|\mathbf{x}_i\right] = 1 - F\left(-Z_i\right) = 1 - F\left(-\beta^T \mathbf{x}_i^T\right)$$

Precisamente esta función de probabilidad acumulativa para distribuciones simétricas como la normal presenta una forma de "S" como la mostrada en el panel c) de la Figura 3.1.

Por otro lado, lastimosamente Z_i es una variable que los investigadores no pueden observar directamente, si bien se puede realizar una inferencia sobre esta al observar

una variable relacionada, como lo es en este caso la variable dummy y_i. A este tipo de variables se les denomina **variable latente**. Es decir, una variable latente es una variable no observable directamente, pero para la cual se puede inferir su valor a partir de una variable relacionada.

Retornando a nuestro problema de clasificación, lo que se desea es encontrar los coeficientes β que en últimas refleja el impacto de cada una de las variables explicativas sobre la probabilidad de que y_i sea uno. Es decir, dada una muestra de tamaño n y sus correspondientes características ($\mathbf{x}_i$) se desean encontrar los $\beta's$ tal que $P\left[Y_i = 1|\mathbf{x}_i\right] = 1 - F\left(-Z_i\right) = 1 - F\left(-\beta^T\mathbf{x}_i^T\right)$ si el individuo es cliente y $P\left[Y_i = 0|\mathbf{x}_i\right] = F\left(-Z_i\right) = F\left(-\beta^T\mathbf{x}_i^T\right)$ si el individuo no es cliente.

A la probabilidad de que la i-ésima observación fuera igual a lo realmente observado se le denomina verosimilitud (en inglés *likelihood*). Así, la verosimilitud para un individuo sea cliente $P\left[y_i = 1|\mathbf{x}_i\right]$ y para los que no son cliente es $P\left[y_i = 0|\mathbf{x}_i\right]$. La función que recoge la verosimilitud (probabilidad) de que ocurra toda la muestra se conoce como la **función de verosimilitud**, es decir:

$$L(\beta) = P(y) = \prod_{\forall y_i=1} P\left(y_i = 1\right) \prod_{\forall y_i=0} P\left(y_i = 0\right)$$

O lo que es equivalente a:

$$L(\beta) = \prod_{\forall y_i=1} 1 - F\left(-Z_i\right) \prod_{\forall y_i=0} F\left(-Z_i\right)$$

Ahora bien, para encontrar los coeficientes β de este tipo de problemas ya no tiene mucho sentido emplear el método de mínimos cuadrados estudiado hasta el momento. De hecho tiene más sentido tratar de encontrar el vector de coeficientes (β) tal que la función de verosimilitud sea maximizada. A este método se le denomina **Método de Máxima Verosimilitud**; el cual implica encontrar los coeficientes tal que el chance de observar la muestra con que se cuenta sea el máximo posible, bajo el supuesto que la información sigue una distribución dada por $F\left(\bullet\right)$.

En otras palabras, los estimadores de máxima verosimilitud corresponden a los coeficientes $\hat{\beta}^{MV}$ tal que:

$$\underset{\beta}{Max}\, L(\beta) = \underset{\beta}{Max}\left[P(y)\right] = \underset{\beta}{Max}\left[\prod_{\forall y_i=1} P\left(y_i = 1\right) \prod_{\forall y_i=0} P\left(y_i = 0\right)\right]$$

En otras palabras:

$$\underset{\beta}{Max}\, L(\beta) = \underset{\beta}{Max}\left[\prod_{\forall y_i=1} 1 - F\left(-Z_i\right) \prod_{\forall y_i=0} F\left(-Z_i\right)\right] \tag{3.4}$$

En general, es mucho más fácil maximizar el logaritmo de la función de máxima verosimilitud (conocida en inglés con el nombre de *log likelihood function*) que la expresión

(3.4). Es decir, en la práctica el problema que se resuelve es:

$$\underset{\beta}{Max} \ln\left(L\left(\beta\right)\right) = \underset{\beta}{Max}\left[\ln\left(P\left(Y\right)\right)\right] \tag{3.5}$$

Dado que la función logarítmica es una transformación monotónica, el vector β que resuelva el problema (3.4) también solucionará el problema (3.5). Es importante resaltar que es difícil encontrar, y en algunos casos no existe, una fórmula que siempre que se aplique dé solución al problema de Máxima Verosimilitud,[8] por tanto el proceso de maximización se resuelve de manera numérica empleando computadores.

Los estimadores de máxima verosimilitud son consistentes y siguen una distribución aproximadamente igual a la normal en muestras grandes. Así mismo, a medida que la muestra crece estos estimadores serán eficientes. Este resultado hace que la inferencia sobre los coeficientes, tanto individual como conjunta, sea igual que lo estudiado para los estimadores MCO.

3.2 Detalles del modelo Logit

En especial, para el modelo *Logit* se supone que la función acumulativa de distribución proviene de una distribución logística[9]. En este caso tendremos que:

$$F\left(Z_i\right) = \frac{e^{-Z_i}}{1+e^{-Z_i}} = \Lambda\left(-Z_i\right)$$

Entonces los estimadores de Máxima Verosimilitud implicarán resolver el siguiente problema

$$\underset{\beta}{Max}\left(\ln\left[\prod_{\forall i\ tal\ que\ y_i=1}\left(1-\Lambda\left(-Z_i\right)\right)\prod_{\forall i\ tal\ que\ y_i=0}\left(\Lambda\left(-Z_i\right)\right)\right]\right)$$

Para el modelo *Logit* los coeficientes β no tienen interpretación por sí solos. En este caso tenemos que:

$$E\left[y_i\left|\mathbf{x}_i\right.\right] = 1 \bullet \left(1-\Lambda\left(-\mathbf{Z_i}\right)\right) + 0 \bullet \Lambda\left(-\mathbf{Z_i}\right) = 1-\Lambda\left(-\mathbf{Z_i}\right)$$

Y, por tanto,

$$\frac{\partial E\left[\mathbf{y_i}\left|\mathbf{x_i}\right.\right]}{\partial \mathbf{X_j}} = -\frac{\partial \Lambda\left(-\mathbf{Z_i}\right)}{\partial \mathbf{X_j}}\beta_j = \Lambda\left(\mathbf{Z_i}\right)\left[1-\Lambda\left(\mathbf{Z_i}\right)\right]\beta_j$$

Noten que para el modelo *Logit* se tiene que el efecto marginal de X_j no es el mismo para todos los individuos. Si la muestra es relativamente grande, puede ser muy complicado analizar el efecto marginal de cada una de las variables explicativas para

[8] Recordemos que en el caso de los estimadores MCO, sí existe una fórmula $(X^TX)^{-1}X^TY$ que garantiza que al aplicarla la suma de los residuos al cuadrado será mínima. Esa fórmula se denomina estimador.

[9] Esta forma de distribución es relativamente parecida a la distribución normal que emplea el modelo *Probit*. Este modelo se presenta en la Sección 3.5.2.

cada uno de los elementos de la muestra. En ese caso una práctica común es calcular todos los efectos marginales para la muestra y calcular el promedio de este. Es decir:

$$\frac{\sum_{i=1}^{n} \Lambda\left(Z_i\right)\left[1-\Lambda\left(Z_i\right)\right]\beta_j}{n}$$

Como se mencionó anteriormente, los computadores pueden encontrar fácilmente por medio de métodos numéricos el vector de coeficientes β^{MV} que resuelve este problema y su correspondiente matriz de varianzas y covarianzas.

3.3 Aplicación del modelo Logit en R

Para realizar un ejemplo práctico emplearemos una base de datos provista por Moro et al. (2014)[10]. Los datos están relacionados con varias campañas de *marketing* directo de una institución bancaria portuguesa. Las campañas de *marketing* se basaron en llamadas telefónicas. A menudo, se requería más de un contacto con el mismo cliente para determinar si el producto (*bank term deposit*) sería adquirido (*yes*) o no.

La base de datos contiene información de 41.188 clientes para 20 variables explicativas y una variable dependiente. Las variables relacionadas con los datos del cliente son:

- `age`: edad (numérico)
- `job`: tipo de trabajo (categórica: "admin.", "blue-collar", "entrepreneur", "housemaid", "management", "retired", "self-employed", "services", "student", "technician", "unemployed", "unknown")
- `marital`: estado civil (categórica: "divorced", "married", "single", "unknown"; nota: "divorced" significa divorciado o viudo)
- `education`: educación (categórica: "basic.4y", "basic.6y", "basic.9y", "high.school", "illiterate", "professional.course", "university.degree", "unknown")
- `default`: incumplimiento: ¿tiene crédito en incumplimiento? (categórica: "no", "yes", "unknown")
- `housing`: vivienda. ¿tiene préstamo de vivienda? (categórica: "no", "yes", "unknown")
- `loan`: préstamo. ¿tiene préstamo personal? (categórica: "no", "yes", "unknown")

Variables relacionadas con el último contacto de la campaña actual:

- `contact`: contacto. Tipo de comunicación de contacto (categórica: "cellular", "telephone")
- `month`: meses. Último mes del año de contacto (categórica: "jan", "feb", "mar", ..., "nov", "dec")
- `day_of_week`: Día de la semana. Último día de contacto de la semana (categórica: "mon", "tue", "wed", "thu", "fri")

[10]La base de datos fue descargada del siguiente enlace: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing.

- `duration`: duración. Duración del último contacto, en segundos (numérico). Nota importante: este atributo afecta en gran medida la variable dependiente (por ejemplo, si duration=0, entonces y = "no"). Sin embargo, la duración no se conoce antes de realizar una llamada. Además, después del final de la llamada, obviamente se conoce y. Por lo tanto, esta variable de entrada solo debe incluirse con fines de referencia y debe descartarse si la intención es tener un modelo predictivo realista.

Otras variables relacionadas con las campañas de mercadeo:

- `campaign`: campaña. número de contactos realizados durante esta campaña y para este cliente (numérica, incluye el último contacto)
- `pdays`: número de días que pasaron después que el cliente fue contactado por última vez desde una campaña anterior (numérica; 999 significa que el cliente no fue contactado previamente)
- `previous`: número de contactos realizados antes de esta campaña y para este cliente (numérica)
- `poutcome`: resultado de la campaña de *marketing* anterior (categórica: "failure", "nonexistent", "success")

Variables asociadas al contexto:

- `emp.var.rate`: tasa de variación del empleo - indicador trimestral (numérica)
- `cons.price.idx`: índice de precios al consumidor - indicador mensual (numérica)
- `cons.conf.idx`: índice de confianza del consumidor - indicador mensual (numérica)
- `euribor3m`: tasa de interés euribor a 3 meses - indicador diario (numérica)
- `nr.employed`: el número de empleados que tenía el banco (numérica)

La variable que se quiere explicar es:

- `y`: ¿ha suscrito el cliente un depósito a plazo? (binario: "yes", "no")

La pregunta de negocio que queremos responder en este caso es: ¿se puede construir un modelo que pueda predecir si un cliente adquirirá o no el producto bancario?

Empecemos por cargar los datos que se encuentran en el archivo `bank-additional-full.csv`[11]

```
# Cargar los datos
datos_originales <- read.csv("./datos/bank-additional-full.csv", sep = ";")
```

Recuerden que la variable `duration` no es útil para hacer la estimación (Ver descripción de las variables arriba). Entonces quitemos esta variable de nuestra muestra empleando la función **subset()** de la base de R.

```
# Eliminar la variable duration
datos <- subset(datos_originales, select = -duration)
```

[11] Los datos se pueden descargar de la página web del libro: http://www.icesi.edu.co/editorial/intro-clasificacion/.

Por otro lado, para la variable `pdays`[12] se tiene que el 96.3 % de los individuos no fueron contactados. Por eso, descartaremos esa variable del análisis.

```
# Eliminar la variable duration
datos <- subset(datos, select = -pdays)
```

Finalmente, la variable `nr.employed`[13] es una variable con poca variabilidad[14] por eso la sacaremos de nuestro análisis.

```
# Eliminar la variable nr.employed
datos <- subset(datos, select = -nr.employed)
```

Ahora asegurémonos que las variables están bien definidas y las que no lo están, las cambiaremos. Adicionalmente, asegurémonos que en la variable dependiente tendremos los "yes" y "no" como positivos (unos) y negativos (ceros), respectivamente. Esto lo podemos hacer con la función **relevel()** del paquete base de R. Con esta función solo necesitamos un primer argumento con la variable a transformar y un segundo argumento (**ref**) que especifica el valor de la variable a transformar que se empleará como el nivel de referencia (igual a 1). Es decir, en este caso:

```
# Chequear la clase de las variables
library(dplyr)
glimpse(datos)
```

```
## Rows: 41,188
## Columns: 18
## $ age            <int> 56, 57, 37, 40, 56, 45, 59, 41, 24, 25, 41, 25, 29, 57,~
## $ job            <chr> "housemaid", "services", "services", "admin.", "service~
## $ marital        <chr> "married", "married", "married", "married", "married", ~
## $ education      <chr> "basic.4y", "high.school", "high.school", "basic.6y", "~
## $ default        <chr> "no", "unknown", "no", "no", "no", "unknown", "no", "un~
## $ housing        <chr> "no", "no", "yes", "no", "no", "no", "no", "no", "yes",~
## $ loan           <chr> "no", "no", "no", "no", "yes", "no", "no", "no", "no", ~
## $ contact        <chr> "telephone", "telephone", "telephone", "telephone", "te~
## $ month          <chr> "may", "may", "may", "may", "may", "may", "may", "may",~
## $ day_of_week    <chr> "mon", "mon", "mon", "mon", "mon", "mon", "mon", "mon",~
## $ campaign       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ previous       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ poutcome       <chr> "nonexistent", "nonexistent", "nonexistent", "nonexiste~
## $ emp.var.rate   <dbl> 1.1, 1.1, 1.1, 1.1, 1.1, 1.1, 1.1, 1.1, 1.1, 1.1, 1.1, ~
## $ cons.price.idx <dbl> 93.994, 93.994, 93.994, 93.994, 93.994, 93.994, 93.994,~
## $ cons.conf.idx  <dbl> -36.4, -36.4, -36.4, -36.4, -36.4, -36.4, -36.4, -36.4,~
```

[12] Esta variable se define como el número de días que pasaron después de que el cliente fue contactado por última vez desde una campaña anterior. Esta variable fue codificada de tal manera que 999 significa que el cliente no fue contactado previamente.

[13] El número de empleado del banco.

[14] La desviación estándar de esta variable es de 72.3 con una media de 5167, lo que implica un coeficiente de variación de 0.014.

```
## $ euribor3m      <dbl> 4.857, 4.857, 4.857, 4.857, 4.857, 4.857, 4.857, 4.857,~
## $ y              <chr> "no", "no", "no", "no", "no", "no", "no", "no", "no", "~
```

```
# Crear la variable dependiente como factor
datos$y <- relevel(as.factor(datos$y), ref = "no")
```

Antes de continuar, no es una buena práctica emplear "." en el nombre de las variables. Miremos el nombre de las variables con el siguiente código:

```
names(datos)
```

```
##  [1] "age"            "job"            "marital"        "education"
##  [5] "default"        "housing"        "loan"           "contact"
##  [9] "month"          "day_of_week"    "campaign"       "previous"
## [13] "poutcome"       "emp.var.rate"   "cons.price.idx" "cons.conf.idx"
## [17] "euribor3m"      "y"
```

Cambiemos los "." por "_" en el nombre de las variables empleando la función **clean_names()** del paquete *janitor*. (Firke, 2023)

```
library(janitor)
datos <- clean_names(datos)
names(datos)
```

```
##  [1] "age"            "job"            "marital"        "education"
##  [5] "default"        "housing"        "loan"           "contact"
##  [9] "month"          "day_of_week"    "campaign"       "previous"
## [13] "poutcome"       "emp_var_rate"   "cons_price_idx" "cons_conf_idx"
## [17] "euribor3m"      "y"
```

Ahora inspeccionemos la proporción de clientes que adquirieron el producto. Esto lo podemos hacer fácilmente en R (R Core Team, 2023) de la siguiente manera:

```
table(datos$y)
```

```
##
##    no   yes
## 36548  4640
```

```
table(datos$y)/nrow(datos) * 100
```

```
##
##       no      yes
## 88.73458 11.26542
```

En este caso tenemos una muestra relativamente balanceada entre los `no` y los `yes`[15].

[15]Esta afirmación es muy discutible. No existe un consenso sobre que umbral debería emplearse para determinar cuándo una muestra se considera desbalanceada, pues puede depender del problema específico y del dominio de aplicación. Sin embargo, algunos autores consideran que una muestra está desbalanceada cuando una clase representa menos del 20 % de las observaciones totales (Ver por ejemplo Chawla et al.

Es importante anotar que si la muestra tiene muy pocos valores de una de las clases de la variable dependiente se puede producir un problema denominado el sesgo por datos desbalanceados. Dicho sesgo se puede solucionar utilizando técnicas de muestreo para equilibrar las clases. Algunas de estas técnicas incluyen el **submuestreo** de la clase mayoritaria o el **sobremuestreo** de la clase minoritaria. Otra opción es utilizar algoritmos de clasificación que sean menos sensibles al desbalance de clases, como el *Random Forest* (Ver Capítulo 8) o el *Support Vector Machine* (Ver Capítulo 9). Además, es importante evaluar el desempeño del modelo utilizando métricas adecuadas para datos desbalanceados, como la **precisión**, **la sensibilidad** y **la especificidad**. Pero siempre será necesario analizar cómo las métricas se pueden ver afectadas por el desbalanceo de la muestra. Cómo solucionar ese problema se discutirá en parte en el Capítulo 10.

Como se anotó en el Capítulo 1, es importante tener una estrategia de **validación cruzada**. Por simplicidad, empleemos el **método de retención** (*holdout method*). Procedamos a crear la muestra de estimación y la de evaluación siguiendo la costumbre de retener el 20% para la evaluación del modelo. Dado que se trata de una muestra de corte trasversal no existe ningún problema en generar este tipo de muestra, primero seleccionemos de manera aleatoria qué observaciones harán parte de cada una de estas muestras.

Hay que recordar que, debido a la naturaleza aleatoria de la muestra que vamos a generar, es importante emplear una semilla para la generación de números aleatorios[16] para garantizar que la muestra generada sea la misma para ti y para nosotros. En R podemos emplear la función **set.seed()** de la base de R para fijar la semilla. Creemos una muestra aleatoria con los índices (número de observación) de las observaciones que harán parte de nuestra muestra de estimación (80% de las observaciones de la muestra original). Esto lo podemos hacer de la siguiente manera:

```
set.seed(123)
# filas que harán parte de la muestra de estimación
est_index <- sample(1:nrow(datos), 0.8 * nrow(datos))
# filas que harán parte de la muestra de evaluación
eval_index <- setdiff(1:nrow(datos), est_index)
```

Ahora extraigamos las correspondientes observaciones para armar las dos muestras.

```
# muestra de estimación
```

(2002), Weiss y Provost (2003) y García et al. (2010)). Sin embargo, otros autores argumentan que el umbral debería ser 10% (Ver por ejemplo Batista et al. (2004), He y Garcia (2009) y Weiss y Provost (2003)). Por ahora daoptaremos arbitrariamente el umbral del 10%.

[16] Los generadores de números aleatorios son empleados por el *software* (como R o cualquier otro) para simular la aleatoriedad que es imposible generar en un computador digital. Es importante tener en cuenta que, en realidad, los números generados por un computador no son verdaderamente aleatorios, ya que se generan utilizando algoritmos deterministas. Estos algoritmos utilizan una "semilla" inicial para producir una secuencia de números aparentemente aleatorios. La calidad de un generador de números aleatorios se evalúa en función de su capacidad para producir secuencias que se asemejen a una secuencia de números verdaderamente aleatorios en términos de distribución y aleatoriedad percibida. Al fijar la semilla, se garantiza que dichos algoritmos siempre generan los mismos números aleatorios.

```
datos_est <- datos[est_index, ]

# muestra de evaluación
datos_eval <- datos[eval_index, ]
```

Antes de continuar, constatemos que las dos muestras siguen estando balanceadas.

```
table(datos_est$y)
```

```
##
##    no   yes
## 29185  3765
```

```
table(datos_est$y)/nrow(datos_est) * 100
```

```
##
##      no     yes
## 88.5736 11.4264
```

```
table(datos_eval$y)
```

```
##
##   no  yes
## 7363  875
```

```
table(datos_eval$y)/nrow(datos_eval) * 100
```

```
##
##       no      yes
## 89.37849 10.62151
```

3.3.1 Estimación del modelo

En este caso $\mathbf{x}_i$ corresponde a las 19 variables explicativas que tenemos[17] y el objetivo nuestro es estimar un modelo *Logit* por medio del método de máxima verosimilitud. De esta manera, el modelo *Logit* para explicar la probabilidad de adquirir el producto está dado por:

$$P\left(y_i = 1|\mathbf{x}_i^T\right) = \Lambda\left(\beta^T \mathbf{x}_i^T\right)$$

donde $\Lambda\left(Z\right) = \frac{e^Z}{1+e^Z}$; es decir, la función de densidad logística. Como se discutió, el vector de coeficientes β puede ser estimado por medio del método de máxima verosimilitud, que equivale a resolver el siguiente problema:

$$\underset{\hat{\beta}}{Max}\, P\left(Y|X\right) = \underset{\hat{\beta}}{Max} \left[\prod_{i=1}^{n} \Lambda\left(\beta^T \mathbf{x}_i^{\mathbf{T}}\right)\right]$$

[17] Recuerden que la variable `duration` no es útil para hacer la estimación.

Este problema es equivalente a resolver lo siguiente:

$$\underset{\hat{\beta}}{Max}\, l\left(Y\,|X\right) = \underset{\hat{\beta}}{Max}\left[\prod_{i=1}^{n} \ln\left[\Lambda\left(\beta^T \mathbf{x_i^T}\right)\right]\right]$$

donde $\ln\left[\bullet\right]$ corresponde al logaritmo natural y $l\left(Y\,|X\right)$ es conocido como el logaritmo de la función de verosimilitud.

En R podemos emplear la función **glm()** del core de R. Esta función estima modelos generalizados lineales que son una generalización de los modelos de regresión lineal. Esta función tiene argumentos muy similares a **lm()**. Debemos expresar el modelo a estimar, los datos y en este caso tenemos el argumento **family** que permite determinar el tipo de modelo a estimar. Para estimar un modelo *Logit* necesitamos que **family = "binomial"**. Estimemos un modelo con todas las variables como explicativas.

```
modelo <- glm(y ~ ., data = datos_est, family = "binomial")
summary(modelo)
```

```
##
## Call:
## glm(formula = y ~ ., family = "binomial", data = datos_est)
##
## Coefficients: (1 not defined because of singularities)
##                                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)                   -1.427e+02  1.015e+01 -14.060  < 2e-16 ***
## age                            1.236e-03  2.355e-03   0.525 0.599736
## jobblue-collar                -1.469e-01  7.651e-02  -1.920 0.054895 .
## jobentrepreneur               -9.625e-03  1.169e-01  -0.082 0.934374
## jobhousemaid                  -9.353e-02  1.447e-01  -0.646 0.518061
## jobmanagement                 -7.601e-03  8.295e-02  -0.092 0.926994
## jobretired                     2.676e-01  1.046e-01   2.559 0.010505 *
## jobself-employed              -4.614e-02  1.132e-01  -0.408 0.683527
## jobservices                   -1.181e-01  8.395e-02  -1.407 0.159574
## jobstudent                     2.744e-01  1.107e-01   2.478 0.013219 *
## jobtechnician                  1.278e-02  6.935e-02   0.184 0.853773
## jobunemployed                  5.209e-02  1.223e-01   0.426 0.670080
## jobunknown                    -1.691e-01  2.309e-01  -0.732 0.464034
## maritalmarried                -4.538e-02  6.522e-02  -0.696 0.486571
## maritalsingle                  1.334e-02  7.466e-02   0.179 0.858153
## maritalunknown                 4.175e-01  3.942e-01   1.059 0.289585
## educationbasic.6y              1.024e-01  1.162e-01   0.881 0.378173
## educationbasic.9y             -1.063e-02  9.189e-02  -0.116 0.907896
## educationhigh.school           2.885e-02  8.919e-02   0.323 0.746322
## educationilliterate            1.105e+00  7.391e-01   1.495 0.134969
## educationprofessional.course   3.504e-02  9.825e-02   0.357 0.721355
## educationuniversity.degree     1.059e-01  8.929e-02   1.186 0.235484
## educationunknown               1.364e-01  1.159e-01   1.177 0.239310
```

```
## defaultunknown               -2.658e-01  6.388e-02  -4.161 3.17e-05 ***
## defaultyes                   -8.621e+00  1.136e+02  -0.076 0.939485
## housingunknown               -3.343e-02  1.310e-01  -0.255 0.798505
## housingyes                   -2.255e-02  4.003e-02  -0.563 0.573201
## loanunknown                         NA         NA      NA       NA
## loanyes                      -1.075e-02  5.504e-02  -0.195 0.845115
## contacttelephone             -6.396e-01  7.083e-02  -9.031  < 2e-16 ***
## monthaug                      2.435e-01  1.072e-01   2.271 0.023148 *
## monthdec                      1.849e-01  1.990e-01   0.929 0.352930
## monthjul                      8.224e-02  9.162e-02   0.898 0.369410
## monthjun                     -4.841e-01  1.023e-01  -4.733 2.21e-06 ***
## monthmar                      1.313e+00  1.194e-01  10.997  < 2e-16 ***
## monthmay                     -5.060e-01  7.361e-02  -6.873 6.29e-12 ***
## monthnov                     -5.920e-01  1.031e-01  -5.739 9.51e-09 ***
## monthoct                     -2.593e-01  1.239e-01  -2.092 0.036400 *
## monthsep                     -9.049e-02  1.315e-01  -0.688 0.491233
## day_of_weekmon               -2.205e-01  6.419e-02  -3.435 0.000593 ***
## day_of_weekthu                8.446e-02  6.155e-02   1.372 0.169968
## day_of_weektue                1.253e-02  6.386e-02   0.196 0.844453
## day_of_weekwed                1.646e-01  6.294e-02   2.614 0.008938 **
## campaign                     -5.211e-02  1.046e-02  -4.981 6.33e-07 ***
## previous                      1.159e-01  5.936e-02   1.953 0.050811 .
## poutcomenonexistent           5.805e-01  9.601e-02   6.046 1.48e-09 ***
## poutcomesuccess               1.734e+00  8.806e-02  19.694  < 2e-16 ***
## emp_var_rate                 -1.230e+00  1.110e-01 -11.085  < 2e-16 ***
## cons_price_idx                1.493e+00  1.055e-01  14.154  < 2e-16 ***
## cons_conf_idx                 1.806e-02  5.547e-03   3.256 0.001128 **
## euribor3m                     3.722e-01  8.406e-02   4.428 9.52e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 23417  on 32949  degrees of freedom
## Residual deviance: 18396  on 32900  degrees of freedom
## AIC: 18496
##
## Number of Fisher Scoring iterations: 10
```

Noten que hay unas variables no significativas. De hecho tenemos 17 posibles variables explicativas, esto implica (sin interacciones) $1{,}31072 \times 10^5$ posibles modelos. Empleemos una estrategia inteligente para encontrar un modelo candidato para ser el mejor modelo[18].

[18]Para una discusión detallada de los algoritmos de selección automática de variables en el contexto de la regresión múltiple ver Alonso (2024). La filosofía de estos algoritmos es similar al caso del modelo *Logit*, de tal manera que ese texto puede ser de gran utilidad para aclarar lo que estamos haciendo en este contexto.

3.3.2 Selección automática de variables explicativas por medio de regresión paso a paso (Stepwise)

Ya conocemos la forma en que, el algoritmo *Stepwise* (en sus versiones adelante, atrás o ambos) puede ayudarnos a encontrar el mejor modelo en un ambiente de regresión múltiple (Ver Sección 3.5.4). Podemos aplicar el mismo algoritmo para el caso del modelo *Logit*. Para esto, podemos emplear la función **step()** del *core* de R. Esta función emplea el AIC o BIC para encontrar el mejor modelo de un objeto de la clase **glm**.

La función **step()** típicamente incluye los siguientes argumentos:

step(object, scope, scale = 0, direction = c("both", "backward", "forward"), trace = 1, keep = NULL, steps = 1000, k = 2)

donde:

- **object**: un objeto de clase **lm** o **gml** que se empleará como el modelo inicial en el proceso de búsqueda.
- **scope**: define el rango de modelos que serán examinados en la búsqueda. Debe ser una fórmula o una lista que contenga componentes superiores e inferiores.
- **direction**: el tipo de búsqueda. Las opciones son "both", "backward" o "forward". El valor por defecto es **direction = "both"**. Si falta el argumento **scope**, la dirección predeterminada será *"backward"*.
- **steps**: El número máximo de pasos a considerar en el algoritmo de búsqueda. El valor predeterminado es 1000.
- **k**: Si **k = log(n)** se emplea el BIC como criterio de selección. El valor por defecto es **k = 2** que corresponde al AIC.

Empleemos esta función para encontrar el mejor modelo con el algoritmo *forward*.

```
# modelo sin variables explicativas
min_model <- glm(y ~ 1, data = datos_est, family = "binomial")

# modelo con todas las potenciales variables
max_model <- glm(y ~ ., data = datos_est, family = "binomial")

# usando forward stepwise para encontrar el mejor modelo
model_forward <- step(min_model, scope = list(lower = min_model, upper =
↪  max_model),
    direction = "forward")
```

El mejor modelo seleccionado es:

```
model_forward$formula
```

```
## y ~ euribor3m + month + poutcome + cons_price_idx + emp_var_rate +
##     contact + day_of_week + campaign + job + default + cons_conf_idx +
##     previous
```

```r
formula_forward <- model_forward$formula
model_forward_sel <- glm(formula_forward, data = datos_est, family =
↪  "binomial")
summary(model_forward_sel)
```

```
##
## Call:
## glm(formula = formula_forward, family = "binomial", data = datos_est)
##
## Coefficients:
##                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)          -1.431e+02  1.014e+01 -14.107  < 2e-16 ***
## euribor3m             3.725e-01  8.404e-02   4.432 9.33e-06 ***
## monthaug              2.506e-01  1.070e-01   2.342 0.019157 *
## monthdec              1.773e-01  1.986e-01   0.893 0.372063
## monthjul              8.407e-02  9.138e-02   0.920 0.357590
## monthjun             -4.845e-01  1.020e-01  -4.752 2.02e-06 ***
## monthmar              1.323e+00  1.192e-01  11.102  < 2e-16 ***
## monthmay             -5.115e-01  7.334e-02  -6.974 3.09e-12 ***
## monthnov             -5.933e-01  1.030e-01  -5.758 8.53e-09 ***
## monthoct             -2.657e-01  1.237e-01  -2.147 0.031807 *
## monthsep             -9.273e-02  1.313e-01  -0.706 0.480107
## poutcomenonexistent   5.840e-01  9.593e-02   6.088 1.15e-09 ***
## poutcomesuccess       1.736e+00  8.800e-02  19.729  < 2e-16 ***
## cons_price_idx        1.498e+00  1.054e-01  14.217  < 2e-16 ***
## emp_var_rate         -1.234e+00  1.109e-01 -11.122  < 2e-16 ***
## contacttelephone     -6.418e-01  7.081e-02  -9.063  < 2e-16 ***
## day_of_weekmon       -2.218e-01  6.413e-02  -3.459 0.000542 ***
## day_of_weekthu        8.578e-02  6.148e-02   1.395 0.162913
## day_of_weektue        1.103e-02  6.378e-02   0.173 0.862648
## day_of_weekwed        1.638e-01  6.289e-02   2.604 0.009221 **
## campaign             -5.162e-02  1.044e-02  -4.944 7.67e-07 ***
## jobblue-collar       -2.014e-01  6.288e-02  -3.203 0.001362 **
## jobentrepreneur      -2.064e-02  1.152e-01  -0.179 0.857848
## jobhousemaid         -1.293e-01  1.380e-01  -0.937 0.348765
## jobmanagement         1.238e-03  8.121e-02   0.015 0.987836
## jobretired            2.513e-01  8.299e-02   3.028 0.002461 **
## jobself-employed     -4.327e-02  1.122e-01  -0.386 0.699776
## jobservices          -1.561e-01  7.965e-02  -1.959 0.050062 .
## jobstudent            2.669e-01  1.012e-01   2.637 0.008358 **
## jobtechnician        -7.333e-03  6.162e-02  -0.119 0.905269
## jobunemployed         2.217e-02  1.207e-01   0.184 0.854298
## jobunknown           -1.541e-01  2.276e-01  -0.677 0.498152
## defaultunknown       -2.691e-01  6.297e-02  -4.274 1.92e-05 ***
## defaultyes           -8.655e+00  1.136e+02  -0.076 0.939252
## cons_conf_idx         1.850e-02  5.529e-03   3.345 0.000822 ***
```

```
## previous             1.171e-01  5.929e-02   1.975 0.048278 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 23417  on 32949  degrees of freedom
## Residual deviance: 18406  on 32914  degrees of freedom
## AIC: 18478
##
## Number of Fisher Scoring iterations: 10
```

Noten que las variables no significativas son parte de las variables dummy creadas en el proceso. Por ahora dejemos el modelo de esta manera.

Hagamos lo mismo para el algoritmo *backward*.

```
model_backward <- step(max_model, scope = list(lower = min_model, upper =
↪  max_model),
    direction = "backward")
```

El mejor modelo seleccionado es:

```
# inspeccionar la fórmula
model_backward$formula
```

```
## y ~ job + default + contact + month + day_of_week + campaign +
##     previous + poutcome + emp_var_rate + cons_price_idx + cons_conf_idx +
##     euribor3m
```

Noten que este modelo no es diferente al anterior.

Finalmente, empleemos el algoritmo *both*.

```
model_both <- step(min_model, scope = list(lower = min_model, upper =
↪  max_model),
    direction = "both")
```

El mejor modelo seleccionado es:

```
# inspeccionar la fórmula
model_both$formula
```

```
## y ~ euribor3m + month + poutcome + cons_price_idx + emp_var_rate +
##     contact + day_of_week + campaign + job + default + cons_conf_idx +
##     previous
```

Este modelo es igual al obtenido por los dos algoritmos anteriores. Ahora regresemos al modelo seleccionado:

```r
summary(model_forward_sel)
```

```
##
## Call:
## glm(formula = formula_forward, family = "binomial", data = datos_est)
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -1.431e+02  1.014e+01 -14.107  < 2e-16 ***
## euribor3m            3.725e-01  8.404e-02   4.432 9.33e-06 ***
## monthaug             2.506e-01  1.070e-01   2.342 0.019157 *
## monthdec             1.773e-01  1.986e-01   0.893 0.372063
## monthjul             8.407e-02  9.138e-02   0.920 0.357590
## monthjun            -4.845e-01  1.020e-01  -4.752 2.02e-06 ***
## monthmar             1.323e+00  1.192e-01  11.102  < 2e-16 ***
## monthmay            -5.115e-01  7.334e-02  -6.974 3.09e-12 ***
## monthnov            -5.933e-01  1.030e-01  -5.758 8.53e-09 ***
## monthoct            -2.657e-01  1.237e-01  -2.147 0.031807 *
## monthsep            -9.273e-02  1.313e-01  -0.706 0.480107
## poutcomenonexistent  5.840e-01  9.593e-02   6.088 1.15e-09 ***
## poutcomesuccess      1.736e+00  8.800e-02  19.729  < 2e-16 ***
## cons_price_idx       1.498e+00  1.054e-01  14.217  < 2e-16 ***
## emp_var_rate        -1.234e+00  1.109e-01 -11.122  < 2e-16 ***
## contacttelephone    -6.418e-01  7.081e-02  -9.063  < 2e-16 ***
## day_of_weekmon      -2.218e-01  6.413e-02  -3.459 0.000542 ***
## day_of_weekthu       8.578e-02  6.148e-02   1.395 0.162913
## day_of_weektue       1.103e-02  6.378e-02   0.173 0.862648
## day_of_weekwed       1.638e-01  6.289e-02   2.604 0.009221 **
## campaign            -5.162e-02  1.044e-02  -4.944 7.67e-07 ***
## jobblue-collar      -2.014e-01  6.288e-02  -3.203 0.001362 **
## jobentrepreneur     -2.064e-02  1.152e-01  -0.179 0.857848
## jobhousemaid        -1.293e-01  1.380e-01  -0.937 0.348765
## jobmanagement        1.238e-03  8.121e-02   0.015 0.987836
## jobretired           2.513e-01  8.299e-02   3.028 0.002461 **
## jobself-employed    -4.327e-02  1.122e-01  -0.386 0.699776
## jobservices         -1.561e-01  7.965e-02  -1.959 0.050062 .
## jobstudent           2.669e-01  1.012e-01   2.637 0.008358 **
## jobtechnician       -7.333e-03  6.162e-02  -0.119 0.905269
## jobunemployed        2.217e-02  1.207e-01   0.184 0.854298
## jobunknown          -1.541e-01  2.276e-01  -0.677 0.498152
## defaultunknown      -2.691e-01  6.297e-02  -4.274 1.92e-05 ***
## defaultyes          -8.655e+00  1.136e+02  -0.076 0.939252
## cons_conf_idx        1.850e-02  5.529e-03   3.345 0.000822 ***
## previous             1.171e-01  5.929e-02   1.975 0.048278 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 23417  on 32949  degrees of freedom
## Residual deviance: 18406  on 32914  degrees of freedom
## AIC: 18478
##
## Number of Fisher Scoring iterations: 10
```

Ahora intentemos crear variables dummy para los factores y evitar que algunas de las categorías no sean significativas. Podemos emplear la función **dummy_cols()** del paquete *fastDummies* (Kaplan, 2023). En el Capítulo 5 de Alonso (2024) se puede encontrar una descripción detallada de cómo funciona esta función.

```
library(fastDummies)
# Crear Dummies de la variable job
dummies_job <- dummy_cols(datos$job, remove_first_dummy = TRUE,
↪  remove_selected_columns = TRUE,
    omit_colname_prefix = TRUE)
# limpiar los nombres de las variables
dummies_job <- clean_names(dummies_job)
# simplificar los nombres
names(dummies_job) <- c("j_blue_collar", "j_entrepreneur", "j_housemaid",
↪  "j_management",
    "j_retired", "j_self_employed", "j_services", "j_student", "j_technician",
    ↪  "j_unemployed",
    "j_unknown")


# Crear Dummies de la variable marital
dummies_marital <- dummy_cols(datos$marital, remove_first_dummy = TRUE,
↪  remove_selected_columns = TRUE)

# limpiar los nombres de las variables
dummies_marital <- clean_names(dummies_marital)

# simplificar los nombres
names(dummies_marital) <- c("m_married", "m_single", "m_unknown")

# Crear Dummies de la variable education
dummies_education <- dummy_cols(datos$education, remove_first_dummy = TRUE,
↪  remove_selected_columns = TRUE)

# limpiar los nombres de las variables
dummies_education <- clean_names(dummies_education)

# simplificar los nombres
```

```
names(dummies_education) <- c("e_basic_6y", "e_basic_9y", "e_high_school",
↪  "e_illiterate",
    "e_professional_course", "e_university_degree", "e_unknown")


# Crear Dummies de la variable default
dummies_default <- dummy_cols(datos$default, remove_first_dummy = TRUE,
↪  remove_selected_columns = TRUE)

# limpiar los nombres de las variables
dummies_default <- clean_names(dummies_default)

# simplificar los nombres
names(dummies_default) <- c("def_unknown", "def_yes")

# Crear Dummies de la variable housing
dummies_housing <- dummy_cols(datos$housing, remove_first_dummy = TRUE,
↪  remove_selected_columns = TRUE)

# limpiar los nombres de las variables
dummies_housing <- clean_names(dummies_housing)

# simplificar los nombres
names(dummies_housing) <- c("h_unknown", "h_yes")

# Crear Dummies de la variable loan
dummies_loan <- dummy_cols(datos$loan, remove_first_dummy = TRUE,
↪  remove_selected_columns = TRUE)

# limpiar los nombres de las variables
dummies_loan <- clean_names(dummies_loan)

# simplificar los nombres
names(dummies_loan) <- c("l_unknown", "l_yes")


# Crear Dummies de la variable contact
dummies_contact <- dummy_cols(datos$contact, remove_first_dummy = TRUE,
↪  remove_selected_columns = TRUE)

# limpiar los nombres de las variables
dummies_contact <- clean_names(dummies_contact)

# simplificar los nombres
names(dummies_contact) <- c("c_telephone")
```

```
# Crear Dummies de la variable months
dummies_month <- dummy_cols(datos$month, remove_first_dummy = TRUE,
↪  remove_selected_columns = TRUE)
# limpiar los nombres de las variables
dummies_month <- clean_names(dummies_month)

# simplificar los nombres
names(dummies_month) <- c("mon_aug", "mon_dec", "mon_jul", "mon_jun",
↪  "mon_mar",
    "mon_may", "mon_nov", "mon_oct", "mon_sep")

# Crear Dummies de la variable dummies_dayWeek
dummies_dayWeek <- dummy_cols(datos$day_of_week, remove_first_dummy = TRUE,
↪  remove_selected_columns = TRUE)
# limpiar los nombres de las variables
dummies_dayWeek <- clean_names(dummies_dayWeek)

# simplificar los nombres
names(dummies_dayWeek) <- c("d_mon", "d_thu", "d_tue", "d_wed")
```

Ahora creemos de nuevo los correspondientes `data.frames` incluyendo las variables dummy creadas.

```
datos_dummies <- subset(datos, select = -c(job, marital, education, default,
↪  housing,
    loan, contact, month, day_of_week))

datos_dummies <- cbind(clean_names(datos_dummies), dummies_job,
↪  dummies_marital,
    dummies_education, dummies_default, dummies_housing, dummies_loan,
↪  dummies_contact,
    dummies_month, dummies_dayWeek)

# Crear la variable dependiente como factor
datos_dummies$y <- relevel(as.factor(datos_dummies$y), ref = "no")

# muestra de estimación
datos_dummies_est <- datos_dummies[est_index, ]

# muestra de evaluación
datos_dummies_eval <- datos_dummies[eval_index, ]
```

Y seleccionemos de nuevo el mejor modelo con esta nueva base de datos y con el algoritmo *forward*.

```r
# modelo sin variables explicativas
min_model_dummies <- glm(y ~ 1, data = datos_dummies_est, family = "binomial")

# modelo con todas las potenciales variables
max_model_dummies <- glm(y ~ ., data = datos_dummies_est, family = "binomial")

# usando forward stepwise para encontrar el mejor modelo
model_forward_dummies <- step(min_model_dummies, scope = list(lower =
↪  min_model_dummies,
    upper = max_model_dummies), direction = "forward")
```

El mejor modelo seleccionado es:

```r
summary(model_forward_dummies)
```

```
##
## Call:
## glm(formula = y ~ euribor3m + poutcome + mon_may + mon_mar +
##     cons_price_idx + cons_conf_idx + c_telephone + emp_var_rate +
##     d_mon + mon_nov + mon_jun + campaign + def_unknown + j_blue_collar +
##     mon_aug + j_retired + j_student + mon_oct + d_wed + e_university_degree +
##     previous + d_thu + j_services, family = "binomial", data = datos_dummies_est)
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -1.455e+02  9.891e+00 -14.706  < 2e-16 ***
## euribor3m            4.037e-01  8.090e-02   4.990 6.04e-07 ***
## poutcomenonexistent  5.792e-01  9.581e-02   6.045 1.49e-09 ***
## poutcomesuccess      1.737e+00  8.788e-02  19.770  < 2e-16 ***
## mon_may             -5.328e-01  5.978e-02  -8.913  < 2e-16 ***
## mon_mar              1.306e+00  1.123e-01  11.630  < 2e-16 ***
## cons_price_idx       1.522e+00  1.026e-01  14.834  < 2e-16 ***
## cons_conf_idx        1.786e-02  4.885e-03   3.656 0.000256 ***
## c_telephone         -6.623e-01  6.681e-02  -9.914  < 2e-16 ***
## emp_var_rate        -1.263e+00  1.085e-01 -11.640  < 2e-16 ***
## d_mon               -2.264e-01  5.529e-02  -4.096 4.21e-05 ***
## mon_nov             -6.423e-01  8.352e-02  -7.690 1.47e-14 ***
## mon_jun             -5.295e-01  8.539e-02  -6.200 5.63e-10 ***
## campaign            -5.077e-02  1.039e-02  -4.885 1.04e-06 ***
## def_unknown         -2.698e-01  6.283e-02  -4.293 1.76e-05 ***
## j_blue_collar       -1.562e-01  5.962e-02  -2.621 0.008771 **
## mon_aug              2.171e-01  8.489e-02   2.557 0.010544 *
## j_retired            2.888e-01  7.974e-02   3.622 0.000293 ***
## j_student            3.068e-01  9.874e-02   3.107 0.001887 **
## mon_oct             -2.911e-01  1.064e-01  -2.736 0.006223 **
## d_wed                1.591e-01  5.373e-02   2.960 0.003074 **
```

```
## e_university_degree  7.210e-02  4.616e-02   1.562 0.118323
## previous             1.143e-01  5.919e-02   1.932 0.053374 .
## d_thu                7.974e-02  5.235e-02   1.523 0.127649
## j_services          -1.150e-01  7.686e-02  -1.496 0.134663
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 23417  on 32949  degrees of freedom
## Residual deviance: 18408  on 32925  degrees of freedom
## AIC: 18458
##
## Number of Fisher Scoring iterations: 6
```

Noten que hay una variable no significativa: `nr_employed`. Eliminémosla.

Veamos este resultado.

```
summary(model_forward_sel_dummies)
```

```
##
## Call:
## glm(formula = model_forward_dummies$formula, family = "binomial",
##     data = datos_dummies_est)
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -1.455e+02  9.891e+00 -14.706  < 2e-16 ***
## euribor3m            4.037e-01  8.090e-02   4.990 6.04e-07 ***
## poutcomenonexistent  5.792e-01  9.581e-02   6.045 1.49e-09 ***
## poutcomesuccess      1.737e+00  8.788e-02  19.770  < 2e-16 ***
## mon_may             -5.328e-01  5.978e-02  -8.913  < 2e-16 ***
## mon_mar              1.306e+00  1.123e-01  11.630  < 2e-16 ***
## cons_price_idx       1.522e+00  1.026e-01  14.834  < 2e-16 ***
## cons_conf_idx        1.786e-02  4.885e-03   3.656 0.000256 ***
## c_telephone         -6.623e-01  6.681e-02  -9.914  < 2e-16 ***
## emp_var_rate        -1.263e+00  1.085e-01 -11.640  < 2e-16 ***
## d_mon               -2.264e-01  5.529e-02  -4.096 4.21e-05 ***
## mon_nov             -6.423e-01  8.352e-02  -7.690 1.47e-14 ***
## mon_jun             -5.295e-01  8.539e-02  -6.200 5.63e-10 ***
## campaign            -5.077e-02  1.039e-02  -4.885 1.04e-06 ***
## def_unknown         -2.698e-01  6.283e-02  -4.293 1.76e-05 ***
## j_blue_collar       -1.562e-01  5.962e-02  -2.621 0.008771 **
## mon_aug              2.171e-01  8.489e-02   2.557 0.010544 *
## j_retired            2.888e-01  7.974e-02   3.622 0.000293 ***
## j_student            3.068e-01  9.874e-02   3.107 0.001887 **
## mon_oct             -2.911e-01  1.064e-01  -2.736 0.006223 **
```

```
## d_wed                1.591e-01  5.373e-02   2.960 0.003074 **
## e_university_degree  7.210e-02  4.616e-02   1.562 0.118323
## previous             1.143e-01  5.919e-02   1.932 0.053374 .
## d_thu                7.974e-02  5.235e-02   1.523 0.127649
## j_services          -1.150e-01  7.686e-02  -1.496 0.134663
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 23417  on 32949  degrees of freedom
## Residual deviance: 18408  on 32925  degrees of freedom
## AIC: 18458
##
## Number of Fisher Scoring iterations: 6
```

Hagamos lo mismo para el algoritmo *backward*.

El mejor modelo es:

```
summary(model_backward_dummies)
```

```
##
## Call:
## glm(formula = y ~ campaign + previous + poutcome + emp_var_rate +
##     cons_price_idx + cons_conf_idx + euribor3m + j_blue_collar +
##     j_retired + j_services + j_student + e_university_degree +
##     def_unknown + c_telephone + mon_aug + mon_jun + mon_mar +
##     mon_may + mon_nov + mon_oct + d_mon + d_thu + d_wed, family = "binomial",
##     data = datos_dummies_est)
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -1.455e+02  9.891e+00 -14.706  < 2e-16 ***
## campaign            -5.077e-02  1.039e-02  -4.885 1.04e-06 ***
## previous             1.143e-01  5.919e-02   1.932 0.053374 .
## poutcomenonexistent  5.792e-01  9.581e-02   6.045 1.49e-09 ***
## poutcomesuccess      1.737e+00  8.788e-02  19.770  < 2e-16 ***
## emp_var_rate        -1.263e+00  1.085e-01 -11.640  < 2e-16 ***
## cons_price_idx       1.522e+00  1.026e-01  14.834  < 2e-16 ***
## cons_conf_idx        1.786e-02  4.885e-03   3.656 0.000256 ***
## euribor3m            4.037e-01  8.090e-02   4.990 6.04e-07 ***
## j_blue_collar       -1.562e-01  5.962e-02  -2.621 0.008771 **
## j_retired            2.888e-01  7.974e-02   3.622 0.000293 ***
## j_services          -1.150e-01  7.686e-02  -1.496 0.134663
## j_student            3.068e-01  9.874e-02   3.107 0.001887 **
## e_university_degree  7.210e-02  4.616e-02   1.562 0.118323
## def_unknown         -2.698e-01  6.283e-02  -4.293 1.76e-05 ***
```

```
## c_telephone         -6.623e-01  6.681e-02  -9.914  < 2e-16 ***
## mon_aug              2.171e-01  8.489e-02   2.557 0.010544 *
## mon_jun             -5.295e-01  8.539e-02  -6.200 5.63e-10 ***
## mon_mar              1.306e+00  1.123e-01  11.630  < 2e-16 ***
## mon_may             -5.328e-01  5.978e-02  -8.913  < 2e-16 ***
## mon_nov             -6.423e-01  8.352e-02  -7.690 1.47e-14 ***
## mon_oct             -2.911e-01  1.064e-01  -2.736 0.006223 **
## d_mon               -2.264e-01  5.529e-02  -4.096 4.21e-05 ***
## d_thu                7.974e-02  5.235e-02   1.523 0.127649
## d_wed                1.591e-01  5.373e-02   2.960 0.003074 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 23417  on 32949  degrees of freedom
## Residual deviance: 18408  on 32925  degrees of freedom
## AIC: 18458
##
## Number of Fisher Scoring iterations: 6
```

Noten que este modelo es exactamente igual al obtenido anteriormente sin la variable no significativa.

Finalmente, empleemos el algoritmo *both*.

```
model_backward_dummies <- step(min_model_dummies, scope = list(lower =
↪  min_model_dummies,
    upper = max_model_dummies), direction = "both")
```

Puedes constatar que este modelo es igual al obtenido por los métodos *forward* y *backward*.

De esta manera tenemos 2 modelos para comparar. Por simplicidad llamémoslos modelo1 y modelo2.

```
modelo1 <- model_forward_sel
modelo2 <- model_forward_sel_dummies
```

Los resultados de estos dos modelos se presentan en el Cuadro 3.1.

Cuadro 3.1. Modelos Logit estimados

	Dependent variable:	
	y	model_forward_dummies
	Modelo 1	Modelo 2
	(1)	(2)
euribor3m	0.372***	0.404***
	(0.084)	(0.081)
monthaug	0.251**	
	(0.107)	
monthdec	0.177	
	(0.199)	
monthjul	0.084	
	(0.091)	
monthjun	−0.485***	
	(0.102)	
monthmar	1.323***	
	(0.119)	
monthmay	−0.511***	
	(0.073)	
monthnov	−0.593***	
	(0.103)	
monthoct	−0.266**	
	(0.124)	
monthsep	−0.093	
	(0.131)	
poutcomenonexistent	0.584***	0.579***
	(0.096)	(0.096)
poutcomesuccess	1.736***	1.737***
	(0.088)	(0.088)
mon_may		−0.533***
		(0.060)
mon_mar		1.306***
		(0.112)
cons_price_idx	1.498***	1.522***
	(0.105)	(0.103)
emp_var_rate	−1.234***	−1.263***
	(0.111)	(0.109)
contacttelephone	−0.642***	
	(0.071)	
day_of_weekmon	−0.222***	
	(0.064)	
day_of_weekthu	0.086	
	(0.061)	
day_of_weektue	0.011	
	(0.064)	
day_of_weekwed	0.164***	
	(0.063)	
d_mon		−0.226***
		(0.055)
mon_nov		−0.642***
		(0.084)
mon_jun		−0.529***
		(0.085)
campaign	−0.052***	−0.051***
	(0.010)	(0.010)
jobblue-collar	−0.201***	
	(0.063)	
jobentrepreneur	−0.021	
	(0.115)	
jobhousemaid	−0.129	
	(0.138)	
jobmanagement	0.001	
	(0.081)	
jobretired	0.251***	
	(0.083)	
jobself-employed	−0.043	
	(0.112)	
jobservices	−0.156*	
	(0.080)	
jobstudent	0.267***	
	(0.101)	
jobtechnician	−0.007	
	(0.062)	
jobunemployed	0.022	
	(0.121)	
jobunknown	−0.154	
	(0.228)	

Para terminar, guarda el espacio de trabajo para ser empleado en los siguientes capítulos.

3.4 Comentarios finales

En esta sección hemos discutido sobre uno de los modelos más usados para la tarea de clasificación. Hemos estudiado cómo generar modelos candidatos a ser los mejores modelos empleando estrategias automáticas de selección de variables. En el siguiente capítulo veremos cómo evaluar e interpretar los resultados de modelos *Logit*. En los próximos capítulos estudiaremos otros modelos estadísticos, así como también métodos de aprendizaje de máquinas.

El modelo *Logit*, así como los otros modelos que estudiaremos, hace parte de la caja de herramientas de los científicos de datos para realizar la tarea de clasificación. *A priori*, es imposible saber si emplear el modelo *Logit* será mejor que los otros y por eso es común que se empleen diferentes aproximaciones y se comparen.

Las aplicaciones de estos métodos de clasificación son muchas en el mundo de los negocios y definitivamente estos modelos son necesarios en la caja de herramientas de un científico de datos.

3.5 Anexos

3.5.1 Demostración de la presencia de Heteroscedasticidad en un Modelo de Probabilidad Lineal

El Modelo de Probabilidad Lineal (MPL) corresponde a un modelo lineal cuya variable dependiente es una dummy. Supongamos que:

$$P\left[\mathbf{y}=1\right]=F\left(\beta^{T}\mathbf{X_i^T}\right)$$

$$P\left[\mathbf{y}=0\right]=1-F\left(\beta^{T}\mathbf{X_i^T}\right)$$

Donde, $\mathbf{X_i}=\left[\begin{array}{cccccc} 1 & \mathbf{X_{2,i}} & \mathbf{X_{3,i}} & \cdots & \mathbf{X_{k,i}} \end{array}\right]_{1\times k}$ corresponde a un vector fila con todas las características del individuo i, y $\beta^{T}=\left[\begin{array}{cccccc} \beta_1 & \beta_2 & \beta_3 & \cdots & \beta_k \end{array}\right]$. Por tanto, el valor esperado de la variable dependiente será:

$$E\left[\mathbf{y_i}\left|\beta^{T}\mathbf{X_i^T}\right.\right]=1\bullet F\left(\beta^{T}\mathbf{X_i^T}\right)+0\bullet\left(1-F\left(\beta^{T}\mathbf{X_i^T}\right)\right)$$

$$E\left[\mathbf{y_i}\left|\beta^{T}\mathbf{X_i^T}\right.\right]=F\left(\beta^{T}\mathbf{X_i^T}\right)$$

El MPL implica suponer que

$$F\left(\beta^{T}\mathbf{X_i^T}\right)=\beta^{T}\mathbf{X_i^T}=\beta_1+\beta_2X_{2,i}+\beta_3X_{3,i}+...+\beta_kX_{k,i}$$

$$F\left(\beta^{T}\mathbf{X_i^T}\right)=\beta^{T}\mathbf{X_i^T}$$

Así, el modelo que describe el comportamiento de la variable dummy será

$$\mathbf{y_i}=E\left[\mathbf{y_i}\left|\beta^{T}\mathbf{X_i^T}\right.\right]+\varepsilon_i$$

$$\mathbf{y_i} = F\left(\beta^T \mathbf{X_i^T}\right) + \varepsilon_i$$
$$Y_i = \beta_1 + \beta_2 X_{2,i} + \beta_3 X_{3,i} + ... + \beta_k X_{k,i} + \varepsilon_i$$

Este modelo lo podemos escribir de forma más compacta de la siguiente manera:

$$\mathbf{y_i} = \beta^T \mathbf{X_i^T} + \varepsilon_i$$

Supongamos que los errores tienen media cero y no están relacionados. Es decir, $E[\varepsilon_i] = 0$ y $E[\varepsilon_i, \varepsilon_j] = 0 \ \forall \ i \neq j$. Además,

$$Y_i = \begin{cases} 1 & \text{con prob. } F \\ 0 & \text{con prob. } 1 - F \end{cases}$$

Por simplicidad escribiremos $F = F\left(\beta^T \mathbf{X_i^T}\right) = \beta^T \mathbf{X_i^T}$. Como $E[\varepsilon_i] = 0$, entonces $Var[\varepsilon_i] = E\left[\varepsilon_i^2\right] - \left[E[\varepsilon_i]\right]^2 = E\left[\varepsilon_i^2\right] - 0 = E\left[\varepsilon_i^2\right]$. Así, debemos determinar a qué es igual el valor esperado del error al cuadrado para determinar si éste es constante o no, para saber si es homoscedástico o heteroscedástico.

Noten que,

$$\mathbf{y_i} = \beta^T \mathbf{X_i^T} + \varepsilon_i = \begin{cases} 0 & \text{con prob. } 1 - F \\ 1 & \text{con prob. } F \end{cases}$$

Esto implica que el término de error tomará únicamente dos posibles valores dado un vector de características $\mathbf{X_i^T}$. Es decir,

$$\varepsilon_i = \begin{cases} -\beta^T \mathbf{X_i^T} & \text{con prob. } 1 - F \\ 1 - \beta^T \mathbf{X_i^T} & \text{con prob. } F \end{cases}$$

Dado que deseamos encontrar la varianza del error y esta corresponde al valor esperado del cuadrado del error ($Var[\varepsilon_i] = E\left[\varepsilon_i^2\right]$), entonces tenemos que tener la expresión del cuadrado del error. Es decir,

$$\varepsilon_i^2 = \begin{cases} \left[-\beta^T \mathbf{X_i^T}\right]^2 & \text{con prob. } 1 - F \\ \left[1 - \beta^T \mathbf{X_i^T}\right]^2 & \text{con prob. } F \end{cases}$$

O lo que es lo mismo,

$$\varepsilon_i^2 = \begin{cases} \beta^T \mathbf{X_i^T} \beta^T \mathbf{X_i^T} & \text{con prob. } 1 - F \\ 1 - 2\beta^T \mathbf{X_i^T} + \beta^T \mathbf{X_i^T} \beta^T \mathbf{X_i^T} & \text{con prob. } F \end{cases}$$

Como se trata de una variable discreta, el valor esperado del error al cuadrado corresponderá a:

$$E\left[\varepsilon_i^2\right] = \left[\beta^T \mathbf{X_i^T} \beta^T \mathbf{X_i^T}\right][1 - F] + \left[1 - 2\beta^T \mathbf{X_i^T} + \beta^T \mathbf{X_i^T} \beta^T \mathbf{X_i^T}\right] F$$
$$E\left[\varepsilon_i^2\right] = \left[\beta^T \mathbf{X_i^T} \beta^T \mathbf{X_i^T}\right] - \left[\beta^T \mathbf{X_i^T} \beta^T \mathbf{X_i^T}\right] F + F - 2\left[\beta^T \mathbf{X_i^T}\right] F + \left[\beta^T \mathbf{X_i^T} \beta^T \mathbf{X_i^T}\right] F$$

$$E\left[\varepsilon_i^2\right] = \left[\beta^T \mathbf{X_i^T} \beta^T \mathbf{X_i^T}\right] + F - 2\left[\beta^T \mathbf{X_i^T}\right] F$$

Como $F = \beta^T \mathbf{X_i^T}$, tenemos que:

$$E\left[\varepsilon_i^2\right] = \left[\beta^T \mathbf{X_i^T} \beta^T \mathbf{X_i^T}\right] + F - 2\left[\beta^T \mathbf{X_i^T}\right] \beta^T \mathbf{X_i^T}$$

$$E\left[\varepsilon_i^2\right] = F - \left[\beta^T \mathbf{X_i^T} \beta^T \mathbf{X_i^T}\right] = \beta^T \mathbf{X_i^T} - \left[\beta^T \mathbf{X_i^T} \beta^T \mathbf{X_i^T}\right]$$

$$E\left[\varepsilon_i^2\right] = \beta^T \mathbf{X_i^T}\left[1 - \beta^T \mathbf{X_i^T}\right]$$

Es decir,

$$Var\left[\varepsilon_i\right] = E\left[\varepsilon_i^2\right] = \beta^T \mathbf{X_i^T}\left[1 - \beta^T \mathbf{X_i^T}\right]$$

$$Var\left[\varepsilon_i\right] = E\left[\mathbf{y_i}\left|\mathbf{X_i}\right.\right]\left[1 - E\left[\mathbf{y_i}\left|\mathbf{X_i}\right.\right]\right]$$

Esto implica que la varianza del error en un modelo de probabilidad lineal depende de cada observación. En otras palabras, por construcción, la varianza del término de error en un modelo de probabilidad lineal no será constante. Por tanto, siempre que la variable dependiente sea una variable dicotómica, un modelo lineal presentará el problema de heteroscedasticidad.

3.5.2 El Modelo Probit

Como se discutió anteriormente, cuando se cuenta con variables dicotómicas como variables dependientes, la probabilidad de observar un valor de uno o cero para una observación depende de la función de distribución que se asuma y más específicamente de su respectiva función de probabilidad acumulativa.

Una función acumulativa que tiene una forma de "S" como la sugerida en el panel c) de la Figura 3.1 es la correspondiente a la distribución normal (ver Figura 3.3). En este caso si suponemos que la función de distribución acumulativa corresponde a una distribución estándar normal tendremos un modelo de elección binario denominado **modelo Probit**, lo cual implica que:

$$F\left(Z_i\right) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-Z_i} e^{-\frac{t^2}{2}} dt = \Phi\left(-Z_i\right) \tag{3.6}$$

Entonces los estimadores de Máxima Verosimilitud implicarán resolver el siguiente problema

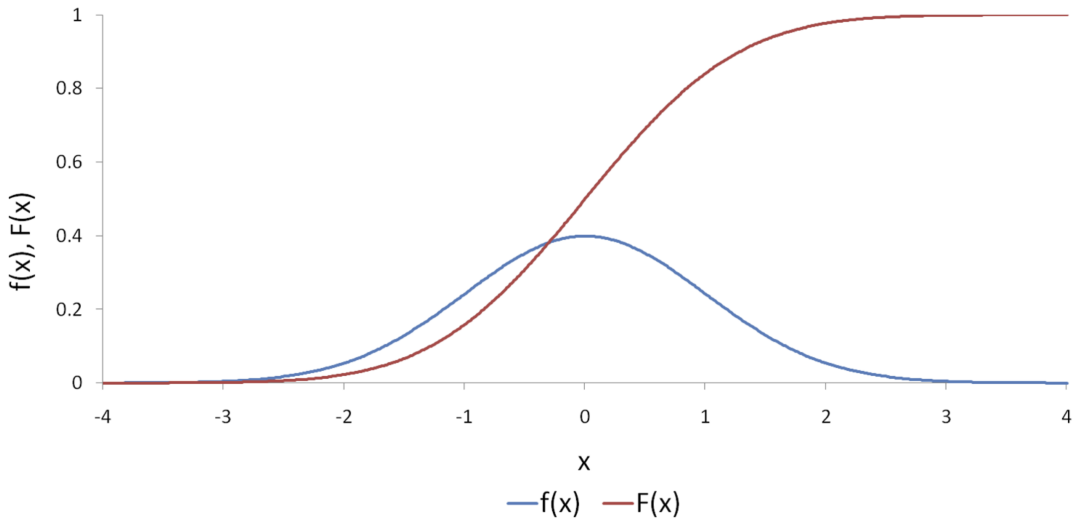
$$\underset{\beta}{Max}\left(\ln\left[\prod_{\forall y_i=1}\left(1 - \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{-Z_i} e^{-\frac{t^2}{2}dt}\right)\prod_{\forall y_i=0}\left(\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{-Z_i} e^{-\frac{t^2}{2}dt}\right)\right]\right)$$

Esto es equivalente al siguiente problema:

$$\underset{\beta}{Max}\left(\ln\left[\prod_{\forall y_i=1}\left(1 - \Phi\left(-Z_i\right)\right)\prod_{\forall y_i=0}\left(\Phi\left(-Z_i\right)\right)\right]\right) \tag{3.7}$$

Como se mencionó anteriormente, los computadores pueden encontrar fácilmente por medio de métodos numéricos el vector de coeficientes β^{MV} que resuelve este problema y su correspondiente matriz de varianzas y covarianzas.

Figura 3.3. Función de distribución estándar normal ($f(x)$) y su correspondiente distribución acumulativa ($F(x)$)



Fuente: elaboración propia.

Un aspecto importante para resaltar en la estimación de un modelo *Probit* es que los coeficientes β no tienen interpretación por si solos. Recuerde que una práctica común para interpretar los coeficientes de un modelo es encontrar la derivada del valor esperado de la variable dependiente con respecto a las variables explicativas. En este caso tenemos que:

$$E\left[y_i \left| \mathbf{x}_i \right.\right] = 1 \bullet \left(1 - \Phi\left(-\mathbf{Z_i}\right)\right) + 0 \bullet \Phi\left(-\mathbf{Z_i}\right) = 1 - \Phi\left(-\mathbf{Z_i}\right)$$

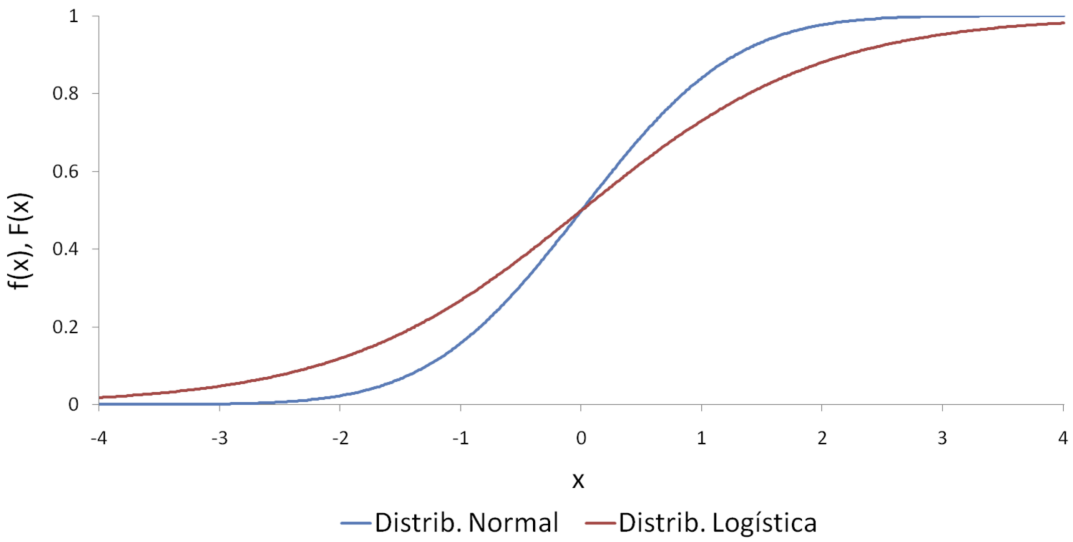
Y, por tanto,

$$\frac{\partial E\left[y_\mathbf{i} \left| \mathbf{x}_{i\mathbf{i}} \right.\right]}{\partial \mathbf{X_j}} = -\frac{\partial \Phi\left(-\mathbf{Z_i}\right)}{\partial \mathbf{X_j}} \beta_j = \frac{1}{\sqrt{2\pi}} e^{-\frac{\mathbf{z_i^2}}{2}} \beta_j = \phi\left(\mathbf{Z_i}\right) \beta_j$$

Esto implica que el efecto de un aumento en una unidad en la variable X_j provocará un cambio de $\phi\left(Z_i\right) \beta_j$ en la probabilidad de que $y_i = 1$. En otras palabras, el efecto marginal de X_j no es el mismo para todos los individuos.

Si la muestra es relativamente grande, puede ser muy complicado analizar el efecto marginal de cada una de las variables explicativas para cada uno de los elementos de la muestra. En ese caso una práctica común es calcular todos los efectos marginales

Figura 3.4. Función de distribución acumulativa normal y logística



Fuente: elaboración propia.

para la muestra y calcular el promedio de este. Es decir:

$$\frac{\sum_{i=1}^{n} \phi\left(Z_i\right) \beta_j}{n}$$

3.5.3 Comentarios sobre los modelos Probit y Logit

La primera diferencia evidente entre emplear un modelo *Probit* o *Logit* es el supuesto sobre la distribución empleada. El primero supone una distribución normal mientras que el segundo implica una distribución logística. De hecho la forma de ambas distribuciones acumulativas de probabilidad es muy parecida (Ver Figura 3.4). Esta forma funcional tan parecida hace que exista una relación entre los coeficientes estimados con cada uno de estos modelos para una misma muestra, en especial se tiene que $\hat{\beta}_{\text{Probit}} \approx 0{,}625\hat{\beta}_{\text{Logit}}$.

Así mismo, los efectos marginales calculados por cada uno de los métodos son relativamente iguales. Por otro lado, no existe ningún método estadístico que permita decidir entre los dos modelos, pues éstos no son comparables.

Finalmente, es importante anotar que el LRI (Ver siguiente capítulo) no es comparable entre modelos *Logit* y *Probit*, pero sí permite comparar modelos *Logit* entre sí o modelos *Probit* entre sí.

3.5.4 Selección Automática de Modelos

En la práctica es común que el científico de datos típicamente se encuentre con problemas en los que se conoce claramente la variable que se quiere explicar (variable dependiente) y un conjunto grande de posibles variables explicativas.

En general, si hay $k-1$ variables independientes potenciales (además del intercepto) que pueden explicar a la variable dependiente, entonces hay $2^{(k-1)}$ subconjuntos de variables explicativas posibles para explicar los cambios en la variable dependiente. En la práctica desconocemos cuál de esos modelos es el correcto. Y por tanto tendríamos que probar todos los posibles modelos para encontrar el correcto. Ahora bien, cuando se cuenta con una teoría este problema no existe. Pero en la mayoría de los casos no contamos con teoría que nos apoye en el proceso. Entonces, por ejemplo, si tenemos 10 ($(k-1)=10$) posibles variables explicativas entonces los posibles modelos serán $2^{(k-1)}=2^{(10)}=1024$. Si se tienen 20 variables candidatas, el número de posibles subconjuntos es de $1{,}048576\times10^6$ (más de un millón de posibles modelos). Si son 25 posibles variables, entonces son $3{,}3554432\times10^7$. Es decir, aproximadamente 33 millones y medio de posibles modelos.

Así, si se cuenta con muchas variables no es viable calcular todos los posibles modelos. No obstante, si las variables son pocas podría ser viable estimar todos los posibles modelo. Es decir, si no tenemos teoría que soporte el modelaje y tenemos muchas variables, entonces tendremos muchos potenciales modelos a evaluar "manualmente".

Aquí presentamos diferentes algoritmos para encontrar el mejor modelo de regresión que se ajuste a unos datos determinados cuando se cuenta con un conjunto relativamente grande de posibles variables explicativas.

La idea de la construcción de modelos por pasos es arribar a un modelo de regresión a partir de un conjunto de posibles variables explicativas basados en un criterio que permita adicionar variables (*stepwise forward regression*) o quitar variables (*stepwise backwards regression*).

Por ejemplo, supongamos que empleamos un criterio como el valor *p* de la prueba de significancia individual de cada variable en el modelo. En el primer caso (*stepwise forward regression*) se parte de un modelo sin variables. Se empieza adicionando al modelo la variable que tenga el mayor valor *p*. De forma gradual se incluye la siguiente variable que tenga el valor *p* más grande, se sigue de esta manera hasta que ya no quede ninguna variable para ingresar que sea significativa[19].

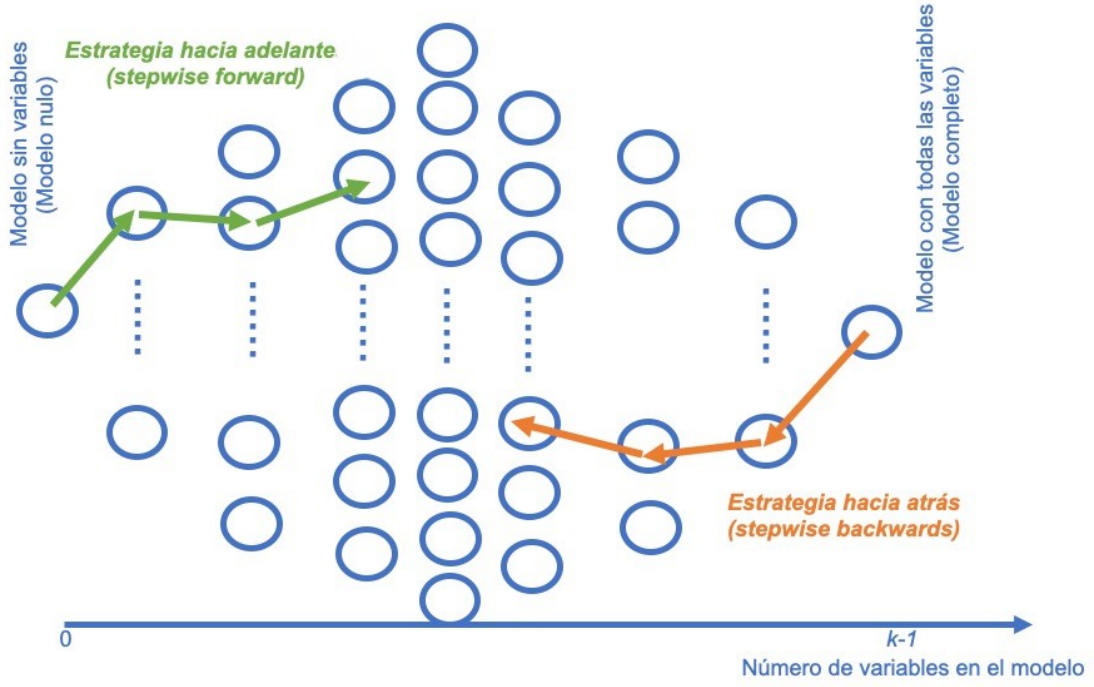
En el segundo caso (*stepwise backwards regression*), se parte del modelo con todas las variables y se empieza a eliminar variables que tenga el valor *p* más bajo. El proceso se repite hasta que no se puedan eliminar variables[20]. Es fácil imaginarse cómo funcionarán ambos métodos si se emplean criterios como el R^2 ajustado o criterios de

[19] Noten que esta aproximación tiene un problema práctico difícil de resolver. Se emplean múltiples pruebas individuales que acumulan el error tipo I. No existe almuerzo gratis, este es el costo de emplear esta aproximación.

[20] Noten que esta aproximación también tiene el problema mencionado para la aproximación *forward*. No existe almuerzo gratis, este es el costo de emplear esta aproximación.

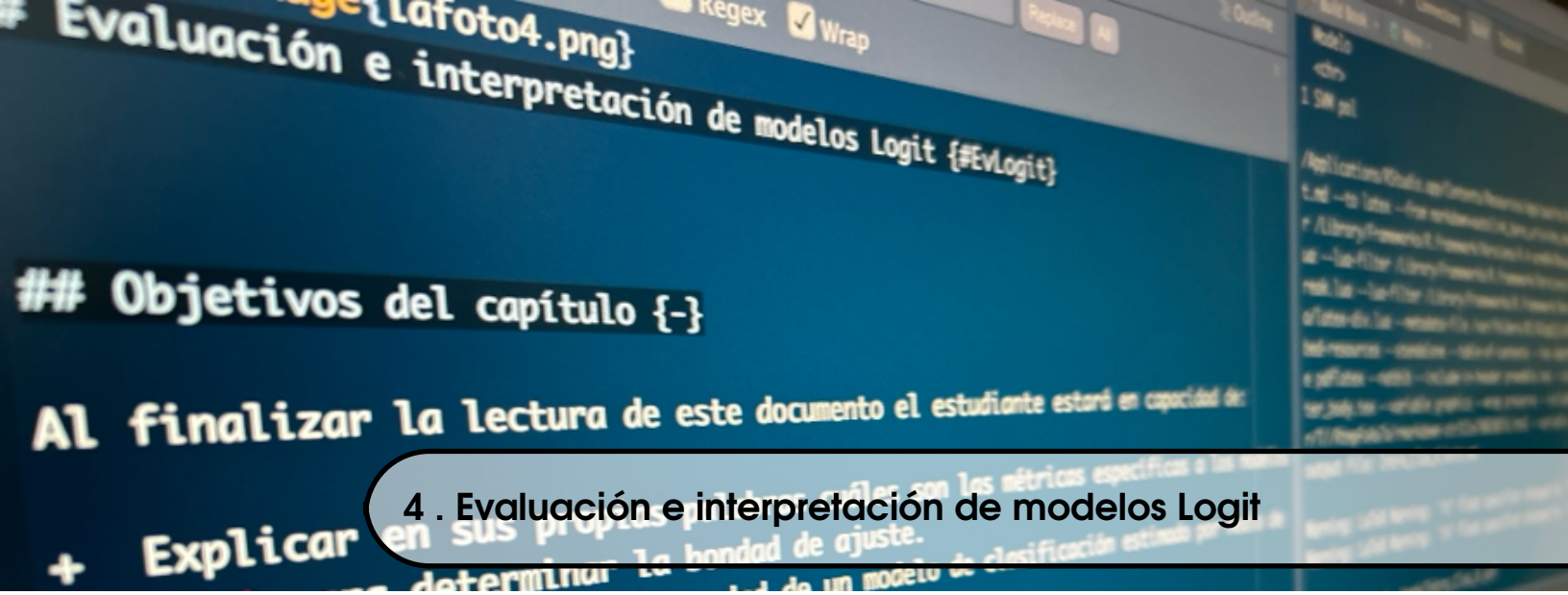
información como **AIC** o **BIC**. La Figura 3.5 muestra de manera esquemática estas dos aproximaciones.

Figura 3.5. Representación de las estrategias stepwise forward y stepwise backwards



Fuente: elaboración propia.

También podemos crear un modelo de regresión a partir de un conjunto de posibles variables explicativas ingresando y eliminando predictores, a este algoritmo se le denomina *both*.



4 . Evaluación e interpretación de modelos Logit

Objetivos del capítulo

Al finalizar la lectura de este documento el estudiante estará en capacidad de:

- Explicar en sus propias palabras cuáles son las métricas específicas a los modelos *Logit* para determinar la bondad de ajuste.
- Calcular métricas de la bondad de un modelo de clasificación estimado por medio de un modelo de regresión logística.
- Interpretar los resultados de un modelo *Logit*.
- Calcular los pronósticos de un modelo de clasificación estimado por medio de un modelo de regresión logística.

4.1 Introducción

En el Capítulo 3 estudiamos cómo estimar un modelo *Logit* en R. Así mismo, se estudió cómo a partir de una conjunto grande de potenciales variables explicativas encontrar el "mejor" modelo, empleando diferentes técnicas de selección automática de modelos. Ahora, es importante discutir cómo comparar diferentes modelos. Como se menciona en el Capítulo 2, la naturaleza de los modelos de clasificación hace que necesitemos herramientas analíticas diferentes a las tradicionales para determinar el ajuste de los modelos. La variable dependiente ya no es continua, sino cualitativa y, en nuestro caso particular, dicotómica. Ya hemos discutido las medidas de ajuste basado en la exactitud del pronóstico (Ver Sección 2.3.2) que son comunes a todos los modelos de clasificación. Además se discutió algunas generalidades para determinar la bondad de un modelo de calificación que genera probabilidades (Ver Sección 2.3.1).

En la siguiente sección nos concentramos en medidas de ajuste específicas al modelo *Logit*. Posteriormente discutiremos cómo calcular en R estas métricas específicas del modelo *Logit*, así como las métricas que se aplican a todos los modelos de clasificación como las discutidas en las secciones 2.3.2 y 2.3.1. Finalmente, tras seleccionar el mejor modelo se discutirá como interpretar los resultados del modelo *Logit*.

4.2 Medidas basadas en el ajuste a la distribución asumida: LRI

Es importante resaltar que en este contexto no tiene mucho sentido emplear una medida como el R^2 para examinar la bondad de ajuste del modelo, pues el método de estimación no pretende minimizar la suma de los errores cuadrados. En este caso se emplea una medida de bondad de ajuste conocido como el **índice de la razón de verosimilitud**, comúnmente conocida como el LRI (por la sigla del término en inglés *Likelihood Ratio Index*). El LRI también es conocido como el R cuadrado de McFadden (*McFadden's R squared*). Esta medida tiene similitudes con el R^2 ajustado al no tener interpretación clara, y no verse afectado por la exclusión o inclusión de variables, por otro lado el LRI se encuentra acotado entre 0 y 1; de tal manera que cuanto más grande el índice, mejor será el ajuste del modelo.

El LRI se calcula de la siguiente manera:

$$LRI = 1 - \frac{\ln L\left(\hat{\beta}\right)}{\ln L_0}$$

Donde $\ln L\left(\hat{\beta}\right)$ corresponde al valor de la función de verosimilitud en su máximo y $\ln L_0$ corresponde al valor del logaritmo de la función de verosimilitud si todos los parámetros fueran cero, excepto el intercepto. Este último valor se puede calcular de la siguiente manera:

$$\ln L_0 = n\left[P\ln\left(P\right) + \left(1-P\right)\ln\left(1-P\right)\right]$$

Donde P es la proporción de observaciones para las cuales la dummy es igual a uno, es decir $P = \left(\#obs = 1\right)/n$. El LRI también es conocido como el Pseudo R^2 de McFadden (McFadden, 1972).

Sin embargo, no es el único estadístico disponible. Cox y Snell (1989) proponen el siguiente indicador similar al R^2 ajustado (conocido como Psuedo R^2):

$$R^2_{Cox\&Snell} = 1 - \left(\frac{L_0}{L\left(\hat{\beta}\right)} \right)^{\frac{2}{n}}$$

Por último, otro denominado Pseudo R^2, es el propuesto por Nagelkerke et al. (1991):

$$R^2_{Nagelkerke} = \frac{1 - \left(\frac{L_0}{L\left(\hat{\beta}\right)} \right)^{\frac{2}{n}}}{1 - (L_0)^{\frac{2}{n}}}$$

Anteriormente se discutió la necesidad de emplear una variable latente (Z_i), no observable para estimar este tipo de modelos. Si bien esa variable no es observable, se puede estimar empleando los β estimados. Es decir,

$$\hat{Z}_i = \hat{\beta}^T \mathbf{x}_i^T$$

Por otro lado, el error ε_i que no es observable no puede ser estimado en este caso, pues no se cuenta con Z_i, para calcular $\hat{\varepsilon}_i = Z_i - \hat{Z}_i$. Esa variable latente puede ser empleada para calcular la probabilidad de que observemos un valor de y_i igual a uno para el individuo i con características registradas en $\mathbf{x}_i$. A esta probabilidad la llamaremos $h_{\hat{\beta}}(\mathbf{x}_i)$ Esto se puede hacer de la siguiente manera:

$$h_{\hat{\beta}}(\mathbf{x}_i) = \Lambda\left(\hat{\beta}^T \mathbf{x}_i^T\right) = \Lambda\left(-\hat{Z}_i\right)$$

4.3 Comparación de los modelos empleando R

Continuando con el ejemplo que iniciamos en el Capítulo 3, procedamos a evaluar los dos modelos que tenemos como candidatos a mejor modelo: el modelo 1 (guardado en el objeto `modelo1`) y el modelo 2 (guardado en el objeto `modelo2`). Carguemos el *working space* que guardamos en el Capítulo 3.

4.3.1 LRI

El LRi se puede encontrar de diferentes formas en R. $\ln L\left(\hat{\beta}\right)$ corresponde al valor de la función de verosimilitud en su máximo y se encuentra fácilmente empleando la función **logLik()** aplicada al objeto de clase **GLM** que contenga el modelo. $\ln L_0$ corresponde al valor del logaritmo de la función de verosimilitud si todos los parámetros fueran cero. Este corresponde al modelo mínimo que definimos arriba. Así, el LRI puede ser calculado manualmente de la siguiente manera:

```
# modelo1

1 - logLik(modelo1)/logLik(min_model)
```

```
## 'log Lik.' 0.213987 (df=36)
```

```
# modelo2

1 - logLik(modelo2)/logLik(min_model_dummies)
```

```
## 'log Lik.' 0.2138849 (df=25)
```

Según este indicador el mejor modelo es el 2. También podemos calcular el LRI (pseudo R^2 de McFadden) y los pseudo R^2 propuestos por Cox y Snell (1989) y por Nagelkerke et al. (1991), usando la función **LogRegR2()** del paquete *descr* (Aquino, 2023).

```
# modelo1
library(descr)
LogRegR2(modelo1)
```

```
## Chi2                 5010.892
## Df                   35
## Sig.                 0
## Cox and Snell Index  0.1410767
## Nagelkerke Index     0.2773341
## McFadden's R2        0.213987
```

```
LogRegR2(modelo2)
```

```
## Chi2                 5008.501
## Df                   24
## Sig.                 0
## Cox and Snell Index  0.1410143
## Nagelkerke Index     0.2772116
## McFadden's R2        0.2138849
```

En todos los 3 estadísticos, el modelo que presenta un mejor ajuste es el `modelo2`. Adicionalmente, es importante aclarar que estas métricas se calculan con la muestra de estimación y no con la de evaluación, pues fue con esta que se estimó el modelo.

Las siguientes métricas se calcularán sobre la muestra de evaluación y no la de estimación para determinar la capacidad del modelo a predecir en presencia de nueva información, tal como lo haría en la vida real.

4.3.2 Curva ROC y AUC

El primer paso es generar predicciones de la probabilidad de observar un positivo para cada individuo; es decir, debemos calcular $h_{\hat{\beta}}(\mathbf{x}_i)$ que lo definimos como

$\Lambda\left(\hat{\beta}^T\mathbf{x}_i^T\right)$. Recuerden que el valor predicho para cada individuo finalmente dependerá de si $h_{\hat{\beta}}(\mathbf{x}_i)$ es mayor o no del punto de corte λ. Estas predicciones la podemos construir con la función **predict()** aplicada al objeto de clase **glm** que tenga el modelo del que se quiere hacer la predicción.

Así mismo, debemos especificar los nuevos datos que se desean emplear para las predicciones (argumento **newdata**). Finalmente, es necesario especificar el tipo de pronóstico. Para el caso de un pronóstico de probabilidad, se emplea el argumento **type = "response"**.

Así, los pronósticos de probabilidades para los individuos de la muestra de evaluación de los dos modelos los podemos encontrar de la siguiente manera:

```
# probabilidades predichas para muestra de evaluación
prob_pred_modelo1 <- predict(modelo1, newdata = datos_eval, type = "response")
prob_pred_modelo2 <- predict(modelo2, newdata = datos_dummies_eval, type =
↪  "response")
```

Ahora ya tenemos todos los elementos necesarios para construir la curva **ROC**. Recuerden que esta curva gráfica la sensibilidad (TPR) y la proporción de falsos negativos (FNR) para cada valor de corte (λ). Existen muchos paquetes para encontrar la curva ROC y el AUC. Nosotros emplearemos el paquete *ROCit* (Khan y Brandenburger, 2020).

La función **rocit()** del paquete *ROCit* (Khan y Brandenburger, 2020) permite calcular el TPR y FPR para diferentes valores de (λ). La función típicamente incluye los siguientes argumentos:

rocit(score, class, negref = NULL, method = "empirical")

donde:

- **score**: Un vector que contiene las probabilidades pronosticadas para cada uno de los individuos ($h_{\hat{\beta}}(\mathbf{x}_i)$).
- **class**: Un vector con los positivos y negativos observados para cada individuo (y_i).
- **method**: El método para el calculo de la **ROC**. Para el caso de modelos *Logit* **{method = "empirical"}**. Este no es el valor por defecto.

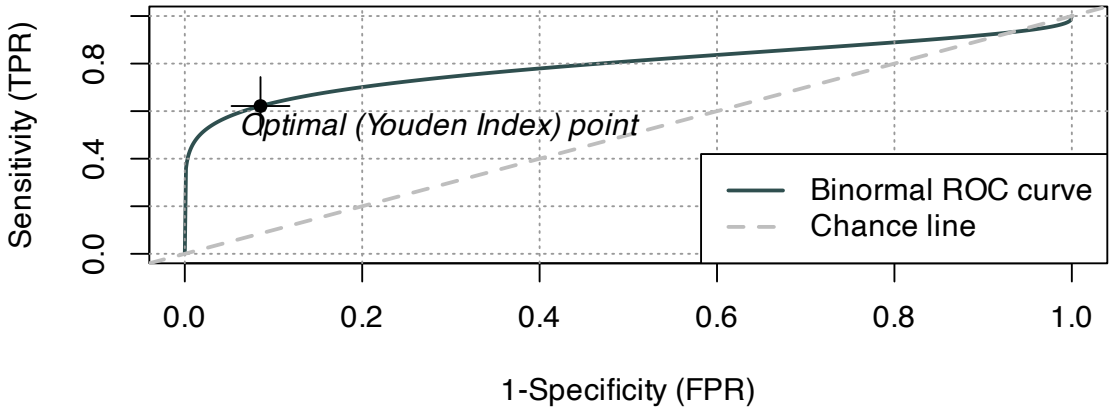
Calculemos la curva **ROC** para el modelo 1.

```
library(ROCit)
roc.modelo1 <- rocit(score = prob_pred_modelo1, class = datos_eval$y, method =
↪  "bin")
```

Y la podemos graficar rápidamente empleando la función **plot()**. La curva **ROC** para el `modelo1` se presenta en la Figura 4.1.

```
plot(roc.modelo1, values = TRUE)
```

Figura 4.1. Curva ROC para el modelo 1



Fuente: elaboración propia.

El área bajo la curva para esta **ROC** (el **AUC**) se puede obtener con la función **summary()** aplicada al objeto que acabamos de crear.

```
summary(roc.modelo1)
```

```
##
##  Method used: binormal
##  Number of positive(s): 875
##  Number of negative(s): 7363
##  Area under curve: 0.7907
```

Ahora calculemos la otra **ROC** (Ver Figura 4.2 y la respectiva **AUC**).

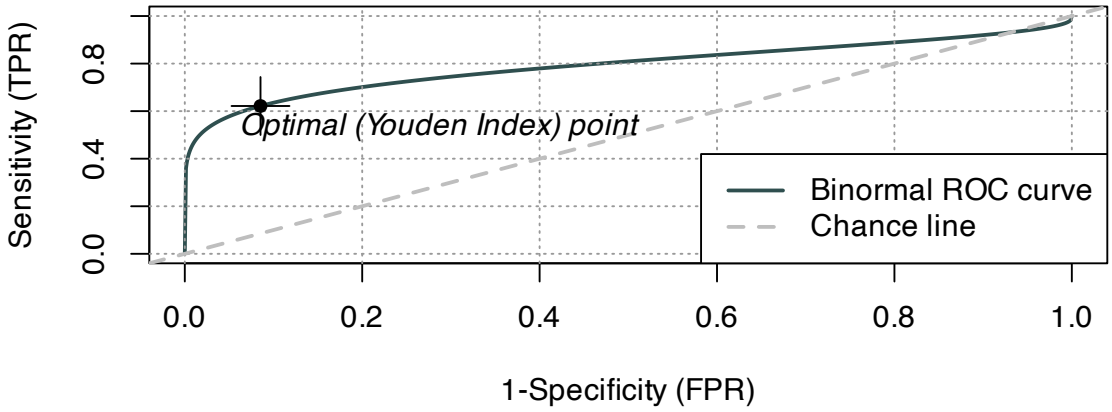
```
roc_modelo2 <- rocit(score = prob_pred_modelo2, class = datos_dummies_eval$y,
↪  method = "bin")

summary(roc_modelo2)
```

```
##
##  Method used: binormal
##  Number of positive(s): 875
##  Number of negative(s): 7363
##  Area under curve: 0.7907
```

Noten que el mejor modelo según esto es el `modelo2` que tiene la **AUC** más alta. Ahora procedamos a escoger el mejor punto de corte para cada uno de los modelos y calculemos las métricas para cada uno de los modelos con su respectivo valor de corte óptimo.

Figura 4.2. Curva ROC para el modelo 2



Fuente: elaboración propia.

4.3.3 Selección del valor de corte óptimo en R

Como se discutió anteriormente, podemos emplear el Índice de Youden (Youden, 1950) (J) para detectar el valor de corte óptimo. Recuerden que

$$J = TPR + TNR - 1$$

Entonces, para construir J para todos los posibles λ podemos emplear la función **measureit()** del paquete *ROCit* (Khan y Brandenburger, 2020). La función típicamente incluye los siguientes argumentos:

measureit(score, class, measure = c("ACC", "SENS"))

donde:

- **score** Un vector que contiene las probabilidades pronosticadas para cada uno de los individuos ($h_{\hat{\beta}}(\mathbf{x}_i)$).
- **class** Un vector con los positivos y negativos observados para cada individuo (y_i).
- **measure** Las métricas que se quieren calcular. Las opciones son: "ACC", "SENS" (Sensibilidad también se puede usar "REC" por Recall), "SPEC" (Especificidad), "PREC" (Precisión), "TPR", "FPR", "TNR", " "FNR" y "FSCR" (Puntaje F_1).

En este caso las métricas de la muestra de entrenamiento para los dos modelos y diferentes valores de λ se pueden calcular de la siguiente manera:

```
metricas_modelo1 <- measureit(score = modelo1$fitted.values, class =
↪  datos_est$y,
    measure = c("SENS", "SPEC", "PREC", "NPV", "ACC", "FSCR", "FPR"))
metricas_modelo2 <- measureit(score = modelo2$fitted.values, class =
↪  datos_est$y,
    measure = c("SENS", "SPEC", "PREC", "NPV", "ACC", "FSCR", "FPR"))
```

Ahora, de los objetos que construimos podemos extraer las métricas que necesitamos para encontrar J y su máximo. Y adicionalmente extraer el correspondiente valor de corte (*Cutoff* en inglés). De la siguiente manera encontramos el valor de corte óptimo.

```
# valor de corte máximo para el modelo 1
opt_pos_modelo1 <- which.max(metricas_modelo1$SENS + metricas_modelo1$SPEC - 1)
v_corte_modelo1 <- metricas_modelo1$Cutoff[opt_pos_modelo1]
v_corte_modelo1
```

```
## [1] 0.1149316
```

```
# valor de corte máximo para el modelo 2
opt_pos_modelo2 <- which.max(metricas_modelo2$SENS + metricas_modelo2$SPEC - 1)
v_corte_modelo2 <- metricas_modelo2$Cutoff[opt_pos_modelo2]
v_corte_modelo2
```

```
## [1] 0.1327713
```

Con este valor de corte podemos proceder a determinar las métricas basadas en la "exactitud" del pronóstico.

4.3.4 Métricas basadas en la "exactitud" del pronóstico en R

Vamos a construir las métricas con el mejor punto de corte para cada modelo, para calcular la matriz de confusión y las métricas podemos emplear la función **confusionMatrix()** del paquete *caret* (Kuhn y Max, 2008). Esta función solo necesita los positivos y negativos que el modelo predice y los observados. Así mismo, es importante especificar cuál es la categoría positiva con el argumento **positive**.

```
# predicción de positivos y negativos

pred_modelo1 <- ifelse(prob_pred_modelo1 > v_corte_modelo1, "yes", "no")
pred_modelo1 <- as.factor(pred_modelo1)

# Cargar paquete
library(caret)


mo_logit_1 <- confusionMatrix(pred_modelo1, datos_eval$y, positive = "yes")

Metrica_modelo1 <- cbind(mo_logit_1$byClass[1], mo_logit_1$byClass[3],
↪  mo_logit_1$byClass[5],
   mo_logit_1$byClass[4], mo_logit_1$overall[1])



# predicción de positivos y negativos
```

```
pred_modelo2 <- ifelse(prob_pred_modelo2 > v_corte_modelo2, "yes", "no")
pred_modelo2 <- as.factor(pred_modelo2)



mo_logit_2 <- confusionMatrix(pred_modelo2, datos_eval$y, positive = "yes")

Metrica_modelo2 <- cbind(mo_logit_2$byClass[1], mo_logit_2$byClass[3],
↪  mo_logit_2$byClass[5],
   mo_logit_2$byClass[4], mo_logit_2$overall[1])

tabla <- rbind(Metrica_modelo1, Metrica_modelo2)
tabla <- as.data.frame(tabla)
row.names(tabla) <- c("Modelo 1", "Modelo 2")
names(tabla) <- c("SENS", "SPEC", "PREC", "NPV", "ACC")
```

En el Cuadro 4.1 se presentan los resultados de las métricas. Se puede observar que el modelo 2 tiene una sensibilidad mayor que el modelo 1. Al tiempo, el valor negativo predictivo (NPV) es mayor en el modelo 2. El modelo 1 tiene una mejor *accuracy* (exactitud). Noten que no es tan fácil decidir basándonos en estas métricas, pues como sabemos existe un compromiso entre estas. En este caso, parece que el modelo 2 es mejor. Pero escoger entre los modelos no es tarea fácil. Esta decisión, al final tendrá que ser una decisión de negocio, pues cada negocio puede determinar que es más deseable para su caso. Por ejemplo, en este caso no parece delicado si se espera que un cliente adquiera el producto y no lo adquiera, pero sería algo más delicado que tengamos potenciales compradores del producto financiero que no se encuentren. En este caso, el modelo 2 podría ser algo mejor, seguiremos trabajando con este.

Cuadro 4.1. Métricas de comparación de modelos para los dos modelos estimados y punto de corte óptimo

	SENS	SPEC	PREC	NPV	ACC
Modelo 1	0.633	0.338	0.338	0.951	0.829
Modelo 2	0.618	0.366	0.366	0.951	0.846

Fuente: elaboración propia.

4.3.5 Matriz de confusión

Ahora calculemos la matriz de confusión del `modelo2`.

```
# Cargar paquete
library(caret)

# Matriz de confusión
```

```
confusionMatrix(pred_modelo2, datos_dummies_eval$y, positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  6427  334
##        yes  936  541
##
##                Accuracy : 0.8458
##                  95% CI : (0.8379, 0.8536)
##     No Information Rate : 0.8938
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3769
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.61829
##             Specificity : 0.87288
##          Pos Pred Value : 0.36628
##          Neg Pred Value : 0.95060
##              Prevalence : 0.10622
##          Detection Rate : 0.06567
##    Detection Prevalence : 0.17929
##       Balanced Accuracy : 0.74558
##
##        'Positive' Class : yes
##
```

Esta matriz de confusión refleja lo ya discutido. Noten que aquí podemos ver que el modelo no es tan bueno como pensábamos. En nuestro caso, el modelo predice correctamente 809 clientes que tomarían el producto financiero y 1196 incorrectamente. Pero el modelo es bueno pronosticando los que no lo tomarían, solo clasifica 66 clientes erróneamente, es decir predice que no tomarán el producto cuando si lo toman.

Finalmente, recuerde que todas las predicciones de la clasificación se encuentran en el objeto `pred_modelo2`. Una vez seleccionado el mejor modelo y adaptado este al negocio, podemos emplear todos los datos para estimar de nuevo el mejor modelo y tenerlo listo para datos de nuevos clientes que van llegando para predecir si estos tomarán o no el producto financiero.

4.4 Interpretación de los resultados del modelo Logit

Los coeficientes β no tienen interpretación directa. Lo único que podemos hacer con ellos, sin ningún cálculo adicional es determinar el signo del efecto de la variable

sobre la probabilidad de observar un positivo.

Para encontrar una interpretación, es importante calcular el efecto marginal promedio de cada variable. Es decir:

$$\frac{\partial E\left[y_{\mathbf{i}} \left| \beta^T \mathbf{x_i^T} \right.\right]}{\partial \mathbf{x_{j,i}}} = \Lambda\left(\beta^T \mathbf{x_i^T}\right)\left[1 - \Lambda\left(\beta^T \mathbf{x_i^T}\right)\right]\beta_j = \left(\frac{e^{\beta^T \mathbf{x_i^T}}}{1 + e^{\beta^T \mathbf{x_i^T}}}\right)\left[1 - \left(\frac{e^{\beta^T \mathbf{x_i^T}}}{1 + e^{\beta^T \mathbf{x_i^T}}}\right)\right] \cdot \beta_j$$

donde $i = 1, 2, ...n$ y $j = 1, 2, ...k$. Este efecto marginal se puede calcular en las medias de las variables explicativas o se evalúa para cada individuo y se calcula su promedio. La segunda aproximación es más común en la actualidad.

Esto se puede hacer fácilmente en R empleando la función **logitmfx()** del paquete *mfx* (Fernihough, 2019). Esta función solo necesita tres argumentos, la fórmula del modelo, los datos y si se desea calcular el efecto marginal en la media de las variables explicativas (**atmean = TRUE**) o el promedio de los efectos marginales de todos los individuos (**atmean = FALSE**). En este caso:

```
# Cargar paquete
library(mfx)
# Calcular los efectos marginales
efectos_marginales <- logitmfx(modelo2$formula, data = datos_dummies_est,
↪  atmean = F)
```

```
print(efectos_marginales)
```

```
## Call:
## logitmfx(formula = modelo2$formula, data = datos_dummies_est,
##     atmean = F)
##
## Marginal Effects:
##                            dF/dx   Std. Err.        z     P>|z|
## euribor3m            0.03199342  0.00645217   4.9586 7.102e-07 ***
## poutcomenonexistent  0.04030968  0.00581560   6.9313 4.170e-12 ***
## poutcomesuccess      0.22401550  0.01607660  13.9343 < 2.2e-16 ***
## mon_may             -0.04058915  0.00437328  -9.2812 < 2.2e-16 ***
## mon_mar              0.14846388  0.01711968   8.6721 < 2.2e-16 ***
## cons_price_idx       0.12065602  0.00854030  14.1278 < 2.2e-16 ***
## cons_conf_idx        0.00141552  0.00038878   3.6409 0.0002717 ***
## c_telephone         -0.04939945  0.00470832 -10.4919 < 2.2e-16 ***
## emp_var_rate        -0.10012371  0.00885917 -11.3017 < 2.2e-16 ***
## d_mon               -0.01727206  0.00405349  -4.2610 2.035e-05 ***
## mon_nov             -0.04416916  0.00493625  -8.9479 < 2.2e-16 ***
## mon_jun             -0.03752536  0.00538670  -6.9663 3.254e-12 ***
## campaign            -0.00402372  0.00082672  -4.8671 1.133e-06 ***
## def_unknown         -0.02036516  0.00450572  -4.5198 6.189e-06 ***
## j_blue_collar       -0.01208323  0.00449583  -2.6877 0.0071956 **
## mon_aug              0.01801637  0.00736639   2.4458 0.0144550 *
```

```
## j_retired            0.02472478  0.00735297   3.3626 0.0007722 ***
## j_student            0.02646280  0.00923536   2.8654 0.0041651 **
## mon_oct             -0.02127139  0.00714989  -2.9751 0.0029292 **
## d_wed                0.01295586  0.00449533   2.8821 0.0039507 **
## e_university_degree  0.00576099  0.00371856   1.5493 0.1213210
## previous             0.00906205  0.00469484   1.9302 0.0535803 .
## d_thu                0.00640267  0.00425693   1.5041 0.1325666
## j_services          -0.00887494  0.00577531  -1.5367 0.1243659
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## dF/dx is for discrete change for the following variables:
##
##  [1] "poutcomenonexistent" "poutcomesuccess"     "mon_may"
##  [4] "mon_mar"             "c_telephone"         "d_mon"
##  [7] "mon_nov"             "mon_jun"             "def_unknown"
## [10] "j_blue_collar"       "mon_aug"             "j_retired"
## [13] "j_student"           "mon_oct"             "d_wed"
## [16] "e_university_degree" "d_thu"               "j_services"
```

Una forma opcional de obtener los efectos marginales y de visualizarlos es el paquete *margins* (Leeper, 2021). Con la función **margins()** podemos calcular los efectos marginales; no obstante, solo nos permite obtener los efectos marginales en la media de las variables explicativas. Esta función requiere dos argumentos principalmente, el primero el objeto con el modelo, y **type**, para indicar el tipo de efecto marginal a estimar, en nuestro caso es "response". Veamos un ejemplo:

```
library(margins)
m <- margins(modelo2, type = "response")
```

```
summary(m)
```

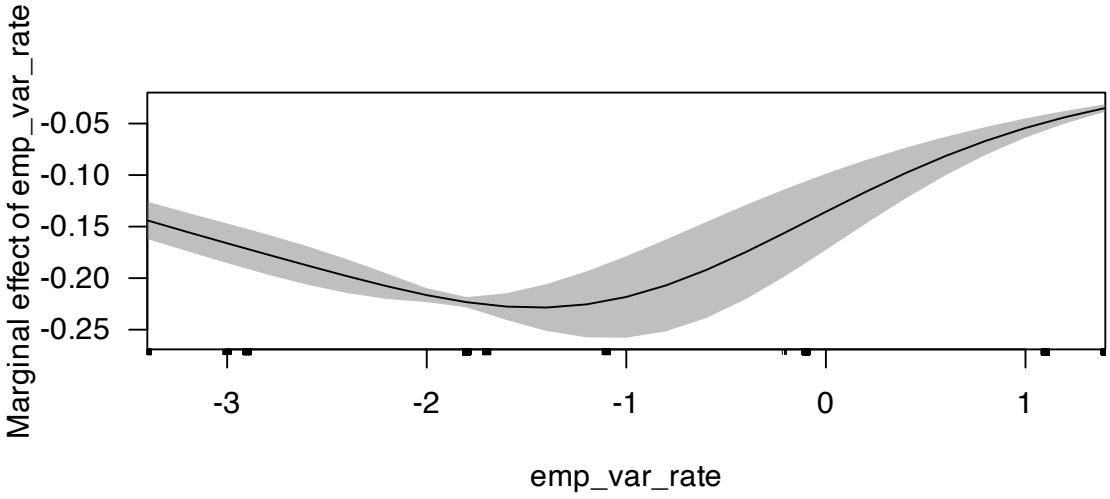
```
##               factor     AME     SE        z      p   lower   upper
##          c_telephone -0.0525 0.0053  -9.9410 0.0000 -0.0628 -0.0421
##             campaign -0.0040 0.0008  -4.8810 0.0000 -0.0056 -0.0024
##         cons_conf_idx  0.0014 0.0004   3.6631 0.0002  0.0007  0.0022
##        cons_price_idx  0.1207 0.0081  14.9869 0.0000  0.1049  0.1364
##                 d_mon -0.0179 0.0044  -4.0968 0.0000 -0.0265 -0.0094
##                 d_thu  0.0063 0.0041   1.5235 0.1276 -0.0018  0.0145
##                 d_wed  0.0126 0.0043   2.9602 0.0031  0.0043  0.0210
##           def_unknown -0.0214 0.0050  -4.2906 0.0000 -0.0311 -0.0116
##  e_university_degree  0.0057 0.0037   1.5619 0.1183 -0.0015  0.0129
##         emp_var_rate -0.1001 0.0086 -11.7004 0.0000 -0.1169 -0.0834
##            euribor3m  0.0320 0.0064   4.9986 0.0000  0.0194  0.0445
##        j_blue_collar -0.0124 0.0047  -2.6206 0.0088 -0.0216 -0.0031
##            j_retired  0.0229 0.0063   3.6234 0.0003  0.0105  0.0353
##           j_services -0.0091 0.0061  -1.4959 0.1347 -0.0211  0.0028
```

```
##            j_student  0.0243 0.0078   3.1086 0.0019  0.0090  0.0396
##              mon_aug  0.0172 0.0067   2.5589 0.0105  0.0040  0.0304
##              mon_jun -0.0420 0.0068  -6.2065 0.0000 -0.0552 -0.0287
##              mon_mar  0.1035 0.0088  11.7353 0.0000  0.0862  0.1208
##              mon_may -0.0422 0.0047  -8.9003 0.0000 -0.0515 -0.0329
##              mon_nov -0.0509 0.0066  -7.6948 0.0000 -0.0639 -0.0379
##              mon_oct -0.0231 0.0084  -2.7362 0.0062 -0.0396 -0.0065
##  poutcomenonexistent  0.0408 0.0060   6.8215 0.0000  0.0291  0.0525
##      poutcomesuccess  0.1734 0.0137  12.6366 0.0000  0.1465  0.2003
##             previous  0.0091 0.0047   1.9322 0.0533 -0.0001  0.0183
```

Como podemos observar, los efectos marginales varían levemente. Para visualizar los efectos marginales podemos usar la función **cplot()** también del paquete *margins* (Leeper, 2021). Esta función necesita de tres argumentos, el primero, al igual que en la función **margins()**, es el objeto que contiene el modelo; el segundo **x** es el nombre de la variable de la cual queremos visualizar los efectos marginales y el tercero **what** se refiere a sí queremos visualizar los efectos marginales o los valores predichos para ciertos valores de la variable, en nuestro caso **what="effect"**. En la Figura 4.3 se presenta el efecto marginal para la variable `emp_var_rate` y el área sombreada indica los respectivos intervalos de confianza al 95%.

```
cplot(modelo2, "emp_var_rate", what = "effect")
```

Figura 4.3. Efecto marginal de la variable seleccionada



Fuente: elaboración propia.

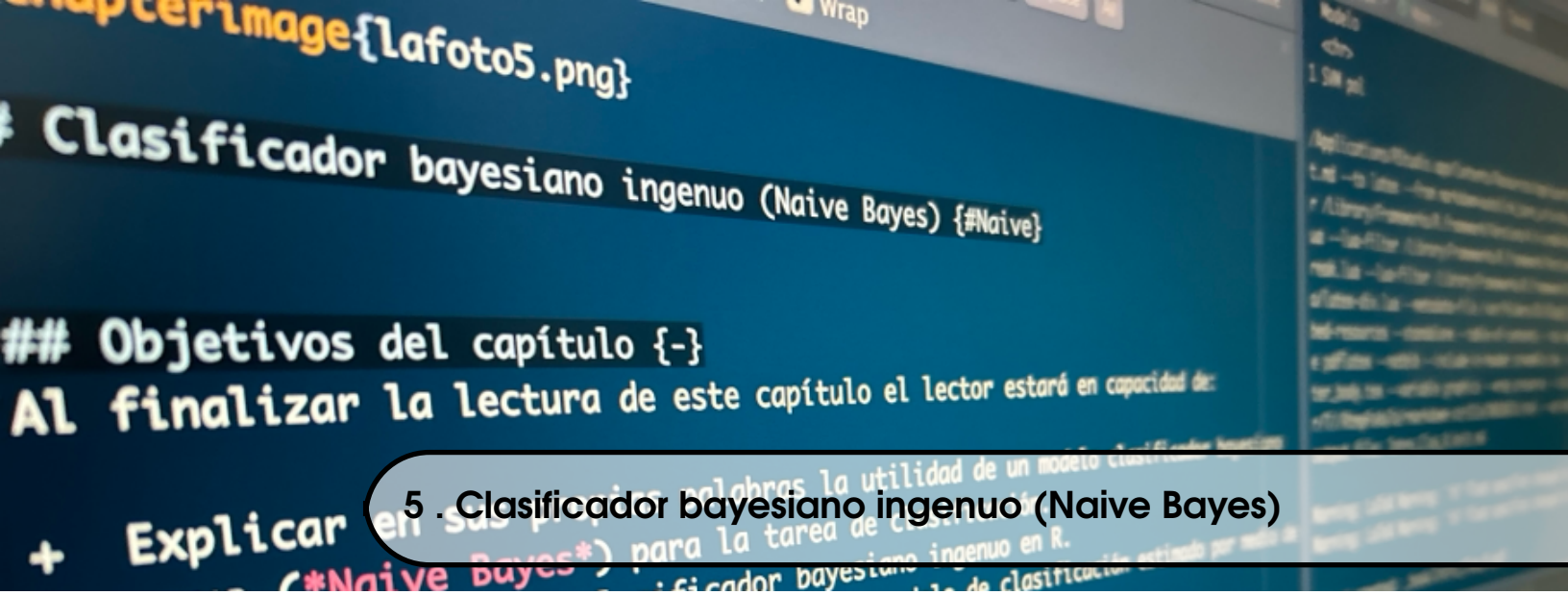
Finalmente, guarda el espacio de trabajo para ser empleado en los siguientes capítulos.

4.5 Comentarios Finales

Antes de terminar es importante decir que la mayoría de los métodos de inferencia usados para el modelo de regresión lineal también aplican para el modelo *Logit*, como por ejemplo, probar cualquier tipo de restricción lineal de la forma $R\beta = C$ por medio de la prueba de *Wald*.

En este capítulo hemos discutido cómo evaluar e interpretar los resultados de modelos *Logit*. En los próximos capítulos estudiaremos otros modelos estadísticos, así como también métodos de aprendizaje de máquinas, que permiten realizar la tarea de clasificación. Este modelo estadístico hace parte de la caja de herramientas para resolver problemas de clasificación en los negocios. *A priori*, es imposible saber si este método será mejor que los de aprendizaje de máquina y por eso es común que se empleen diferentes aproximaciones y se comparen.

Las aplicaciones de estos métodos de clasificación son muchas en el mundo de los negocios y definitivamente estos modelos son necesarios en la caja de herramientas de un científico de datos.



5 . Clasificador bayesiano ingenuo (Naive Bayes)

Objetivos del capítulo

Al finalizar la lectura de este capítulo el lector estará en capacidad de:

- Explicar en sus propias palabras la utilidad de un modelo clasificador bayesiano ingenuo (*Naive Bayes*) para la tarea de clasificación.
- Emplear un modelo clasificador bayesiano ingenuo en R.
- Calcular métricas de la bondad de un modelo de clasificación estimado por medio de un modelo *Naive Bayes*.
- Encontrar las predicciones de clasificación de un modelo *Naive Bayes*.

5.1 Introducción

El clasificador bayesiano ingenuo (también conocido como clasificador *Naive Bayes* o solamente *Naive Bayes*) es otro método estadístico que se basa en el teorema de Bayes para realizar la tarea de clasificación bajo el supuesto (ingenuo) de que las variables explicativas (o predictoras) son independientes entre sí.

El teorema de Bayes permite conocer cuál es la probabilidad de observar uno de los posibles valores de la variable aleatoria A (A_i)[1] dado que se observó el evento B. Esta probabilidad se escribe como[2] $P(A_i|B)$. Esta probabilidad se conoce como la probabilidad *a posteriori* o probabilidad condicionada.

El teorema de Bayes implica que:

$$P(A_i|B) = \frac{P(A_i)P(B|A_i)}{P(B)} \tag{5.1}$$

donde, $P(A_i)$ es la probabilidad no condicionada de que ocurra el evento A_i. $P(B)$ es la probabilidad de observar el evento B. Y $P(B|A_i)$ es la probabilidad de observar el evento B dado que ocurre A_i.

Intuitivamente, se puede ver cómo es posible adaptar el teorema de Bayes para el caso de la clasificación con dos clases. En este caso solo existen dos posibles valores para la variable A: $A = 1$ y $A = 0$. Adicionalmente, se puede considerar que B corresponde a las características del individuo a clasificar. Entonces, por ejemplo la probabilidad de que para un individuo se observe un uno será igual a:

$$P(A = 1|B) = \frac{P(A = 1)P(B|A = 1)}{P(B)}$$

Este método de clasificación empleará las frecuencias observadas en la muestra para calcular, por individuo, tanto la probabilidad de ser igual a uno como ser igual a cero para las características determinadas del individuo. Aquella categoría con mayor probabilidad será la asignada por este algoritmo.

En lo que resta de este capítulo discutiremos formalmente este método y demostraremos cómo emplear R para implementar el clasificador bayesiano ingenuo.

5.2 Detalles del modelo Naive Bayes

Definamos C_j como las categorías de y, nuestra variable de interés. Adicionalmente, para encontrar la probabilidad de observar una determinada categoría C_j emplearemos las características recogidas en las p variables explicativas: $(X_1, X_2, X_3, \ldots, X_p)$. Entonces, para cada individuo nuestro objetivo será calcular:

$$P(y = C_j|X_1, X_2, X_3, \ldots, X_p)$$

[1] A_i puede tomar cualquier valor del conjunto $(A_1, A_2, ..., A_i, ..., A_n)$ con probabilidad $P(A_i)$. $P(A_i)$ se conoce como la probabilidad *a priori* o no condicionada de que ocurra el evento A_i.

[2] Esto se lee como la probabilidad de que ocurra A_i dado B.

En otras palabras, lo que deseamos encontrar es la probabilidad de pertenecer a cierta clase C_j dado ciertos valores de nuestras variables explicativas. Aplicando el teorema de Bayes, para una realización de las variables $X_1, X_2, X_3, ..., X_p$ tenemos:

$$P(y = C_j|x_1, x_2, \ldots, x_p) = \frac{P(y = C_j)P(x_1, x_2, \ldots, x_p|y = C_j)}{P(x_1, x_2, \ldots, x_p)}$$

Ahora, suponiendo (de manera ingenua) que las variables explicativas $(x_1, x_2, \ldots, x_p)$ son independientes para las categorías C_j tendremos que:

$$P(x_1, x_2, \ldots, x_p|y = C_j) = P(x_1|y = C_j)P(x_2|y = C_j) \ldots P(x_p|y = C_j)$$

y además la independencia de las variables explicativas implicará:

$$P(x_1, x_2, \ldots, x_p) = P(x_1)P(x_2) \ldots P(x_p)$$

Finalmente, entonces el teorema de Bayes en este contexto y con el supuesto de independencia implicará:

$$P(y = C_j|x_1, x_2, \ldots, x_p) = \frac{P(y = C_j) \prod_{j=1}^{p} P(x_j|y = C_j)}{P(x_1)P(x_2)...P(x_p)}$$

Noten que para un individuo el denominador será igual para cualquier categoría que adopte la variable y. Así, si se desea encontrar la categoría C_j con la mayor probabilidad para un individuo, esto equivale a comparar solamente los numeradores. O dicho de otra manera, la probabilidad condicional es proporcional al numerador. Es decir,

$$P(y = C_j|x_1, x_2, \ldots, x_p) \propto P(y = C_j) \prod_{j=1}^{p} P(x_j|y = C_j) \tag{5.2}$$

Entonces, la idea de este algoritmo es asignar la categoría C_j al individuo i si esta tiene la mayor probabilidad de ocurrencia. Es decir,

$$C_j = argmax_{C_j} P(y = C_j) \prod_{j=1}^{p} P(x_j|y = C_j) \tag{5.3}$$

Evaluaremos el producto de la probabilidad no condicionada de observar cada categoría de y y la probabilidad condicional de observar cada uno de los valores de las variables explicativas del individuo i condicionado a la correspondiente categoría. Y se escoge la categoría que tenga dicho producto más alto.

La probabilidad no condicionada $P(y = C_j)$ no es más que dividir el total de observaciones en la clase j entre el total de observaciones de todas las clases. El término $\prod_{j=1}^{p} P(x_j|y = C_j)$ se requiere de las distribuciones de probabilidad condicional $f(X_j|y = C_j)$ para $j = 1, 2, 3, .., p$. Habitualmente se asume normalidad si la variable es continua y distribución multinomial si es discreta.

5.3 Aplicación en R del modelo Naive Bayes

Para nuestro trabajo en R, seguiremos empleando los mismos datos que empleamos en el Capítulo 3 y Capítulo 4 (Ver detalles de cada uno de los pasos para construir la base de datos en el Capítulo 3). Carga el *working space* que guardaste en el Capítulo 4.

Recuerda que contamos con un objeto que contiene la muestra de estimación `datos_dummies_est` y otro con la muestra de evaluación `datos_dummies_eval`.

Para aplicar el modelo *Naive Bayes* en R, vamos a usar el paquete *e1071* (Meyer et al., 2023) , existen otros paquetes como *naivebayes* (Majka, 2019) que también permite hacer la estimación del modelo. La función **naiveBayes()**, realiza los pasos anteriormente descritos para el clasificador bayesiano ingenuo. Esta función requiere argumentos similares a la función **glm()** estudiada en el Capítulo 3, el primer argumento debe ser la fórmula y el segundo los datos. Veamos un ejemplo empleando todas las variables en el objeto :

```
library(e1071)
# Estimar el modelo Naive Bayes
model_naive <- naiveBayes(y ~ ., data = datos_dummies_est)

print(model_naive)
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##       no      yes
## 0.885736 0.114264
##
## Conditional probabilities:
##      age
## Y         [,1]      [,2]
##   no  39.90269  9.889701
##   yes 41.03373 13.918015
##
##      campaign
## Y         [,1]     [,2]
##   no  2.638444 2.914564
##   yes 2.030013 1.659482
##
##      previous
## Y          [,1]      [,2]
```

```
##   no  0.1322255 0.4066817
##   yes 0.5003984 0.8794576
##
##      poutcome
## Y        failure nonexistent    success
##   no  0.10008566  0.88665410 0.01326024
##   yes 0.12934927  0.67808765 0.19256308
##
##      emp_var_rate
## Y           [,1]     [,2]
##   no   0.2440055 1.484797
##   yes -1.2249668 1.618526
##
##      cons_price_idx
## Y         [,1]      [,2]
##   no  93.60406 0.5597387
##   yes 93.36144 0.6752099
##
##      cons_conf_idx
## Y          [,1]     [,2]
##   no  -40.59526 4.395171
##   yes -39.89065 6.102165
##
##      euribor3m
## Y         [,1]     [,2]
##   no  3.805837 1.640795
##   yes 2.125707 1.741372
##
##      j_blue_collar
## Y          [,1]      [,2]
##   no  0.2360116 0.4246367
##   yes 0.1362550 0.3431047
##
##      j_entrepreneur
## Y           [,1]      [,2]
##   no  0.03659414 0.1877664
##   yes 0.02788845 0.1646751
##
##      j_housemaid
## Y           [,1]      [,2]
##   no  0.02604077 0.1592593
##   yes 0.02177955 0.1459824
##
##      j_management
## Y           [,1]      [,2]
##   no  0.07102964 0.2568787
```

```
##   yes 0.07224436 0.2589265
##
##      j_retired
## Y           [,1]      [,2]
##   no  0.03536063 0.1846928
##   yes 0.09614874 0.2948343
##
##      j_self_employed
## Y           [,1]      [,2]
##   no  0.03477814 0.1832206
##   yes 0.03213811 0.1763902
##
##      j_services
## Y           [,1]      [,2]
##   no  0.09957170 0.2994332
##   yes 0.06932271 0.2540358
##
##      j_student
## Y           [,1]      [,2]
##   no  0.01668665 0.1280967
##   yes 0.06002656 0.2375676
##
##      j_technician
## Y          [,1]      [,2]
##   no  0.1647079 0.3709231
##   yes 0.1553785 0.3623132
##
##      j_unemployed
## Y           [,1]      [,2]
##   no  0.02360802 0.1518271
##   yes 0.03187251 0.1756840
##
##      j_unknown
## Y            [,1]       [,2]
##   no  0.008017817 0.08918410
##   yes 0.008499336 0.09181142
##
##      m_married
## Y          [,1]      [,2]
##   no  0.6130204 0.4870673
##   yes 0.5426295 0.4982456
##
##      m_single
## Y          [,1]      [,2]
##   no  0.2717492 0.4448689
##   yes 0.3450199 0.4754379
```

```
##
##      m_unknown
## Y            [,1]       [,2]
##   no  0.001816001 0.04257659
##   yes 0.002921647 0.05398041
##
##      e_basic_6y
## Y           [,1]      [,2]
##   no  0.05718691 0.2322034
##   yes 0.03984064 0.1956106
##
##      e_basic_9y
## Y          [,1]      [,2]
##   no  0.1518246 0.3588569
##   yes 0.1011952 0.3016271
##
##      e_high_school
## Y          [,1]      [,2]
##   no  0.2341271 0.4234593
##   yes 0.2223108 0.4158541
##
##      e_illiterate
## Y             [,1]       [,2]
##   no  0.0003083776 0.01755828
##   yes 0.0007968127 0.02822037
##
##      e_professional_course
## Y          [,1]      [,2]
##   no  0.1269145 0.3328829
##   yes 0.1280212 0.3341578
##
##      e_university_degree
## Y          [,1]      [,2]
##   no  0.2869625 0.4523517
##   yes 0.3598938 0.4800328
##
##      e_unknown
## Y           [,1]      [,2]
##   no  0.04019188 0.1964124
##   yes 0.05577689 0.2295209
##
##      def_unknown
## Y           [,1]      [,2]
##   no  0.22288847 0.4161912
##   yes 0.09508632 0.2933731
##
```

```
##      def_yes
## Y             [,1]       [,2]
##   no  0.0001027925 0.01013832
##   yes 0.0000000000 0.00000000
##
##      h_unknown
## Y           [,1]      [,2]
##   no  0.02388213 0.1526846
##   yes 0.02390438 0.1527716
##
##      h_yes
## Y          [,1]      [,2]
##   no  0.5198218 0.4996155
##   yes 0.5397078 0.4984870
##
##      l_unknown
## Y           [,1]      [,2]
##   no  0.02388213 0.1526846
##   yes 0.02390438 0.1527716
##
##      l_yes
## Y          [,1]      [,2]
##   no  0.1525441 0.3595537
##   yes 0.1500664 0.3571839
##
##      c_telephone
## Y          [,1]      [,2]
##   no  0.3907144 0.4879189
##   yes 0.1715803 0.3770654
##
##      mon_aug
## Y          [,1]      [,2]
##   no  0.1504883 0.3575555
##   yes 0.1359894 0.3428228
##
##      mon_dec
## Y            [,1]       [,2]
##   no  0.002535549 0.05029122
##   yes 0.018061089 0.13319008
##
##      mon_jul
## Y          [,1]      [,2]
##   no  0.1771458 0.3817986
##   yes 0.1434263 0.3505536
##
##      mon_jun
```

```
## Y          [,1]      [,2]
##   no  0.1311633 0.3375846
##   yes 0.1235060 0.3290608
##
##      mon_mar
## Y           [,1]       [,2]
##   no  0.00729827 0.08511905
##   yes 0.06002656 0.23756760
##
##      mon_may
## Y          [,1]      [,2]
##   no  0.3528182 0.4778549
##   yes 0.1909695 0.3931172
##
##      mon_nov
## Y           [,1]      [,2]
##   no  0.10083947 0.3011212
##   yes 0.08924303 0.2851321
##
##      mon_oct
## Y           [,1]      [,2]
##   no  0.01134144 0.1058924
##   yes 0.06401062 0.2448044
##
##      mon_sep
## Y            [,1]       [,2]
##   no  0.008771629 0.09324691
##   yes 0.054448871 0.22693142
##
##      d_mon
## Y          [,1]      [,2]
##   no  0.2081891 0.4060198
##   yes 0.1824701 0.3862828
##
##      d_thu
## Y          [,1]      [,2]
##   no  0.2075381 0.4055511
##   yes 0.2278884 0.4195260
##
##      d_tue
## Y          [,1]      [,2]
##   no  0.1957855 0.3968110
##   yes 0.1984064 0.3988528
##
##      d_wed
## Y          [,1]      [,2]
```

```
##   no  0.1956484 0.3967058
##   yes 0.2071713 0.4053332
```

Realicemos la predicción, empleando la función **predict()** aplicada al objeto de clase **naiveBayes** que tiene el modelo del que se quiere hacer la predicción. Esta es la misma función que empleamos en el Capítulo 4, la diferencia es que en este caso el argumento **type** debe ser **"class"**, de esta forma, sobre la muestra de evaluación las predicciones son:

```
# Calcular las predicciones del modelo para la muestra de evaluación
pred_modelo_naive <- predict(model_naive, newdata = datos_dummies_eval, type =
↪  "class")

# ver las 10 primeras predicciones
head(pred_modelo_naive, 10)
```

```
##  [1] no no no no no no no no no no
## Levels: no yes
```

Como podemos observar, ya no nos arroja una probabilidad sino la categoría a la que pertenece la observación, en nuestro caso "yes" o "no".

5.4 Bondad de ajuste del modelo

Ahora calculemos las métricas de evaluación derivadas de la matriz de confusión, usando la función **confusionMatrix()** del paquete *caret* (Kuhn y Max, 2008) como lo realizamos en el Capítulo 4:

```
library(caret)
# Calcular métricas del modelo Naive Bayes
mo_naive <- confusionMatrix(pred_modelo_naive, datos_dummies_eval$y, positive =
↪  "yes")
print(mo_naive)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  6620  462
##        yes  743  413
##
##                Accuracy : 0.8537
##                  95% CI : (0.8459, 0.8613)
##     No Information Rate : 0.8938
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3251
##
```

```
##  Mcnemar's Test P-Value : 7.257e-16
##
##             Sensitivity : 0.47200
##             Specificity : 0.89909
##          Pos Pred Value : 0.35727
##          Neg Pred Value : 0.93476
##              Prevalence : 0.10622
##          Detection Rate : 0.05013
##    Detection Prevalence : 0.14033
##       Balanced Accuracy : 0.68555
##
##        'Positive' Class : yes
##
```

Ahora podemos compararlo con lo obtenido con el mejor modelo Logit identificado en el Capítulo 4. En ese caso teníamos:

```
confusionMatrix(pred_modelo2, datos_dummies_eval$y, positive = "yes")$table
```

```
##           Reference
## Prediction   no  yes
##        no  6427  334
##        yes  936  541
```

Las otras métricas estudiadas del mejor modelo Logit y el *Naive Bayes* se reportan en el Cuadro 5.1. La medida de *accuracy* del modelo de *Naive Bayes* (85.4 %) es más alto comparado con el modelo Logit (84.6 %); sin embargo, clasificando el valor positivo "yes", no es muy bueno. Se observa que clasifica mal a 743[3] observaciones en la clase "yes", que realmente no fueron "yes". Noten que un problema similar se presenta en el modelo Logit (con 936 falsos positivos que corresponden al 63.4 % de los positivos que predice el modelo). Veamos las métricas para los dos modelos vistos hasta el momento.

Cuadro 5.1. Métricas de comparación de modelos para modelo Logit y Naive Bayes

	Accuracy	Sensitivity	Specificity	Precision	F1
Modelo Logit	0.846	0.618	0.873	0.366	0.460
Naive Bayes	0.854	0.472	0.899	0.357	0.407

Fuente: elaboración propia.

Para terminar, guarda el espacio de trabajo para ser empleado en los siguientes capítulos.

[3] Esto representa el 64.3 % de todos los positivos predichos por el modelo *Naive Bayes*.

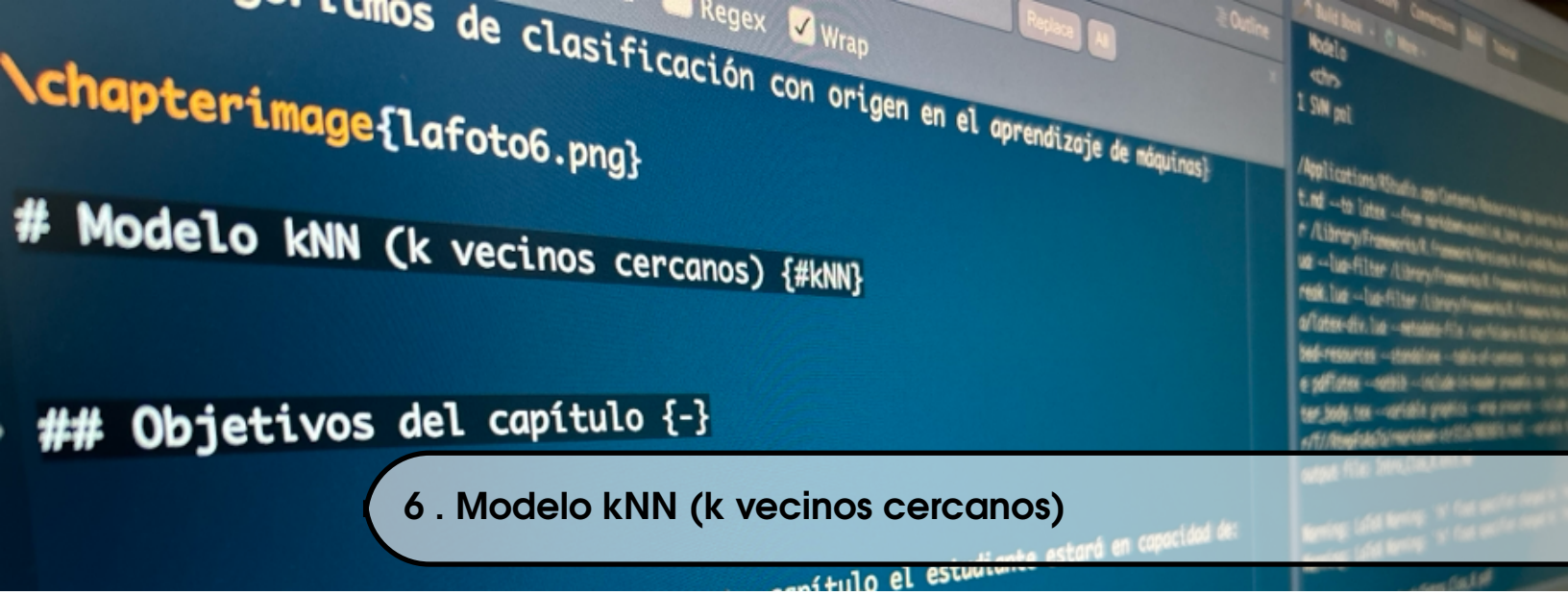
5.5 Comentarios Finales

En este Capítulo hemos aprendido otro método estadístico para realizar la tarea de clasificación. En los próximos capítulos estudiaremos métodos de aprendizaje de máquinas que también sirven para realizar la tarea de clasificar. Como se mencionó anteriormente, *a priori*, es imposible saber si este método será mejor que el estadístico u otros de aprendizaje de máquina. Por eso, es una buena práctica emplear diferentes aproximaciones y compararlas para obtener el mejor modelo para responder la pregunta de negocio.

Las aplicaciones de estos métodos de clasificación son muchas en el mundo de los negocios y definitivamente estos modelos son necesarios en la caja de herramientas de un científico de datos.

Parte III

Algoritmos de clasificación con origen en el aprendizaje de máquinas



6 . Modelo kNN (k vecinos cercanos)

Objetivos del capítulo

Al finalizar la lectura de este capítulo el estudiante estará en capacidad de:

- Explicar en sus propias palabras la utilidad de un modelo kNN.
- Emplear un modelo kNN en R.
- Calcular métricas de la bondad de un modelo de clasificación estimado por medio de un modelo kNN.
- Encontrar los pronósticos de clasificación de un modelo kNN.

6.1 Introducción

Como lo hemos discutido a lo largo de este libro, una de las tareas del *Business Analytics* más comunes es clasificar individuos en varias categorías o clases. Los modelos de clasificación que comúnmente se emplean tienen dos orígenes: modelos estadísticos y modelos de aprendizaje de máquinas[1]. En los capítulos anteriores discutimos modelos estadísticos. Ahora nos concentraremos en una técnica de inteligencia artificial conocida como kNN.

El aprendizaje automático (o inteligencia artificial) es la ciencia que emplea algoritmos que permiten a los computadores aprender, sin ser explícitamente programados (Ng, 2014). En la inteligencia artificial, los algoritmos son expuestos a una muestra (conocida como *training set* o muestra de entrenamiento) para aprender patrones en los datos que permitan replicar ese patrón de comportamiento en una nueva muestra (muestra de evaluación).

Hay dos tipos de aprendizaje automático: el supervisado y el no supervisado. En el aprendizaje supervisado, los datos de entrenamiento contienen la variable objetivo y otras variables. El algoritmo emplea diferentes iteraciones y formas funcionales para encontrar el patrón en el que las variables en la muestra se asocian a la variable objetivo (pueden ser más de una). En este caso, tenemos datos de entrenamiento etiquetados previamente por un "experto" tanto para la variable objetivo y_i, así como de las demás variables ($X_{1,i}, X_{2,i}, \cdots, X_{n,i}$). En este sentido, la meta de los modelos de aprendizaje supervisado es predecir con las variables $X_{1,i}, X_{2,i}, \cdots, X_{n,i}$ la variable objetivo (y_i). La variable objetivo puede ser una variable cuantitativa (continua o discreta) o puede ser cualitativa (clases). En el segundo caso, cuando y_i corresponde a clases, el problema que resuelve el modelo de aprendizaje supervisado será uno de clasificación.

Por otro lado, en los modelos de aprendizaje no supervisado no se cuenta con una variable objetivo etiquetada. Se cuenta con observaciones de muchas variables ($X_{1,i}, X_{2,i}, \cdots, X_{p,i}$) y el objetivo es descubrir factores no observados, estructuras, o una representación más simple de los datos. Noten que estos modelos son útiles para:

- Resumir información como lo hace una aproximación matemática tradicional como el PCA,
- Clasificar observaciones, como la aproximación estadística del clustering jerárquico.
- Encontrar asociaciones o leyes de co-ocurrencia en el análisis de canasta (Ver Alonso y Arboleda (2024) para una discusión de la tarea de encontrar reglas de asociación).

En este Capítulo nos dedicaremos a un algoritmo de aprendizaje supervisado conocido como **k vecinos más próximos** (kNN por su sigla del inglés *k-nearest neighbors*).

La idea de este algoritmo, a diferencia del modelo Logit, no implica calcular parámetros (como los βs) y, por eso, se conoce como un modelo no paramétrico[2]. La

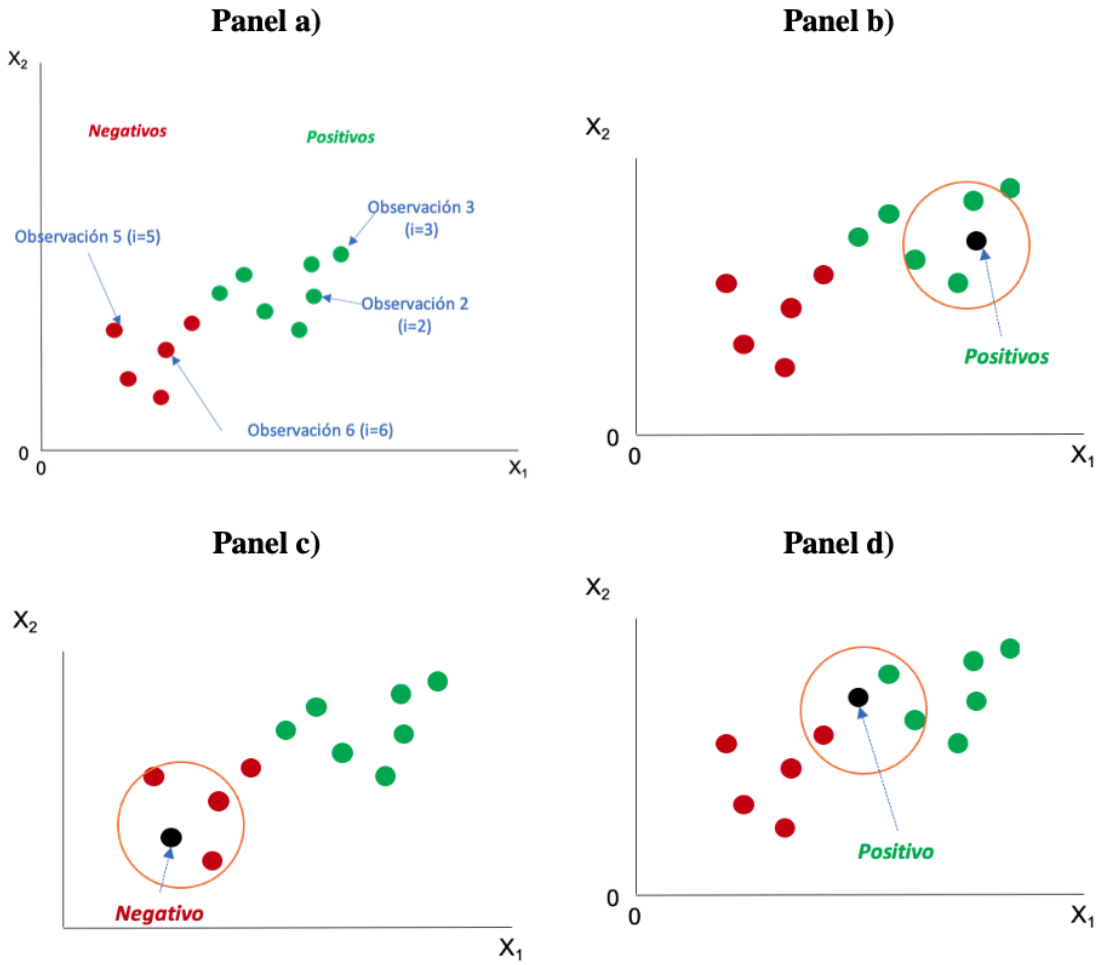
[1] También conocido como aprendizaje automático o aprendizaje automatizado. Este término viene del inglés *machine learning*.

[2] El modelo *Naive Bayes* también es un modelo no paramétrico.

idea detrás de este método es encontrar la probabilidad de que un elemento (i) pertenezca a la clase j (C_j) empleando la información contenida en las variables $X_{1,i}, X_{2,i}, \cdots, X_{p,i}$. Para encontrar dicha probabilidad no es necesario suponer una distribución (como si lo hace el modelo Logit).

El algoritmo kNN implica buscar k "vecinos" cercanos para cada elemento (i) que puedan "votar" cuál debería ser la clase del elemento i. Por ejemplo, supongamos que contamos con dos clases: compra el producto (positivo) y no lo compra (negativo) y tenemos dos variables: nivel de ingresos $(X_{1,i})$ y los años de educación $(X_{2,i})$. (Ver Figura 6.1 panel a).

Figura 6.1. Funcionamiento del algoritmo kNN



Fuente: elaboración propia.

Ahora supongamos que definimos $k = 3$. Es decir, estaremos empleando 3 "vecinos" para votar. El algoritmo irá punto por punto (observación por observación) buscando

sus vecinos y por "votación" aprenderá cuál debería ser la clase de la observación. Consideremos una observación i como la que se presenta con un punto negro en la Figura 6.1 panel b). El algoritmo "olvidará" por un momento que conoce la clase real de la observación i, después buscará 3 vecinos ($k = 3$) mas cercanos. En este caso, resaltados en el circulo naranja en la Figura 6.1 panel b). Ahora los tres vecinos "votarán". Los tres "votos" de los vecinos concuerdan y esto hará que el algoritmo asigne a esta observación la clase de compra el producto (positivo). En el panel c) de la Figura 6.1 se presenta un caso similar, pero todos los votos implicarán que el algoritmo "aprende" que ese individuo es un "no comprador" (negativo). Finalmente, en el panel c) de la Figura 6.1 se presenta un caso en el que el algoritmo encuentra 2 "votos" por positivo y uno por negativo, por mayoría el algoritmo escoge rotular esta observación como un positivo. El algoritmo hace esto para todas las observaciones de la muestra de entrenamiento.

De esta aproximación intuitiva quedan claras las siguientes características del algoritmo kNN:

- Para ejecutar este algoritmo se tendrá que determinar una forma de medir la distancia entre los puntos para determinar cuáles observaciones están más cerca.
- Para calcular las distancias, será necesario tener variables expresada numéricamente (y no como factores).
- Es necesario tener un número de vecinos impar para evitar empates en la "votación".
- k no puede ser un múltiplo del número de clases.
- Las unidades en que se midan las variables ($X_{1,i}, X_{2,i}, \cdots, X_{p,i}$) pueden afectar la medida de distancia y, por eso, será necesario estandarizar los datos[3].
- Si la muestra es grande, este algoritmo puede tomar mucho tiempo encontrando los k vecinos próximos.

En las siguientes secciones de este capítulo veremos cómo escoger k y cómo realizar este algoritmo en R, pero antes se presenta una sección con el detalle técnico del algoritmo (esta sección puede ser omitida si el lector no se siente cómodo con la matemática detrás del algoritmo).

6.2 Detalle técnico

El modelo *Logit* (Ver Capítulo 3) es un modelo paramétrico que intenta encontrar la probabilidad que una observación i tome el valor de uno dada las características de las variables explicativas. kNN es un modelo no paramétrico para encontrar dichas probabilidades. kNN emplea k vecinos para calcular la probabilidad de la siguiente manera:

$$\hat{p}_{k,j}(\mathbf{x}_i) = \hat{P}_k(Y = j \mid \mathbf{x}_i) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(\mathbf{x}_i, \mathcal{D})} I(y_i = j) \tag{6.1}$$

[3]Los dos métodos de escalar datos más empleados son la normalización y la estandarización. La normalización generalmente significa re-escalar los valores en un rango entre $[0, 1]$. La estandarización generalmente significa que los datos re-escalados tienen una media de 0 y una desviación estándar de 1.

En otras palabras, la probabilidad estimada por el algoritmo para cada clase j para la i-ésima observación es la proporción de k vecinos de i con dicha clase (j).

Entonces, para crear un clasificador, el algoritmo asigna a i la clase con la más alta probabilidad estimada. Es decir,

$$\hat{C}_k(\mathbf{x}_i) = \underset{j}{\operatorname{argmax}} \ \hat{p}_{kj}(\mathbf{x}_i) \tag{6.2}$$

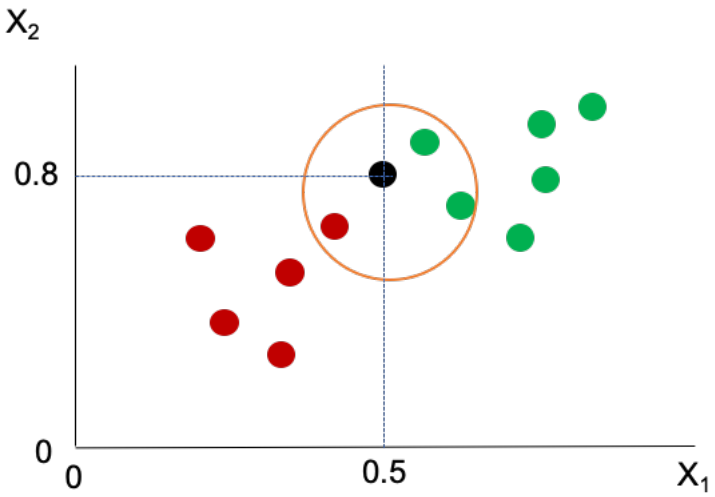
Esto es lo mismo que decir que asignamos la clase a i con la mayor cantidad de observaciones ("votos") en los k vecinos más cercanos. Si por casualidad se llegase a la situación en la que al menos dos clases tienen la mayor probabilidad, el algoritmo kNN asignará una clase al azar entre aquellas con la probabilidad más alta.

En el caso binario, que venimos estudiando, esto se convierte en:

$$\hat{C}_k(\mathbf{x}_i) = \begin{cases} 1 & \hat{p}_{k,0}(\mathbf{x}_i) > 0{,}5 \\ 0 & \hat{p}_{k,0}(\mathbf{x}_i) \leqslant 0{,}5 \end{cases} \tag{6.3}$$

Como se mencionó anteriormente, si la probabilidad para la clase 0 y 1 es igual, se asignará al azar la clase. Una forma de evitar este empate, y, por tanto, la asignación de la clase de manera aleatoria, es emplear k impares y que no sean múltiplos del número de clases.

Figura 6.2. Ejemplo de asignación de clase con el algoritmo kNN



Fuente: elaboración propia.

En la Figura 6.2 se presenta un ejemplo de este algoritmo con $k = 3$. En ese caso, para la observación i (con $X_{1,i} = 0{,}5$ y $X_{2,i} = 0{,}8$, es decir, $\mathbf{x}_i = (0{,}5, 0{,}8)$) las probabilidades serán:

$$\hat{p}_{3,\mathbf{x}_i}(x_{1,i} = 0{,}8, x_{2,i} = 0{,}5) = \hat{P}_3(y_i = 1 \mid x_{1,i} = 0{,}8, x_{2,i} = 0{,}5) = \frac{2}{3}$$

$$\hat{p}_{3,\mathbf{x}_i}(x_{1,i} = 0{,}8, x_{2,i} = 0{,}5) = \hat{P}_3(y_i = 0 \mid x_{1,i} = 0{,}8, x_{2,i} = 0{,}5) = \frac{1}{3}$$

y, por tanto,

$$\hat{C}_3(x_{1,i} = 0{,}8, x_{2,i} = 0{,}5) = 1$$

Adicionalmente, es importante mencionar que es común seleccionar los vecinos más cercanos por medio de la distancia euclidiana. Es decir,

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{=r1}^{p} (X_{r,i} - X_{r,j})^2} \tag{6.4}$$

Finalmente, en la práctica está el problema de seleccionar k. Un k pequeño, le permite al algoritmo identificar patrones muy sutiles, pero esos patrones pueden capturar mucho ruido en los datos (el componente aleatorio), mientras que un k grande reduce el ruido pero puede perder esa especificidad que da el vecindario pequeño.

Lo que es claro, es que el k óptimo depende de la muestra. Algunos autores como Lall y Sharma (1996) sugieren emplear $k = \sqrt{n}$. Pero hoy en día lo más común es calcular el algoritmo para diferentes tamaños de k y comparar el comportamiento del algoritmo en la muestra de evaluación con diferentes métricas como las discutidas en el Capítulo 4. Por ejemplo, se puede seleccionar el K que maximice la *accuracy* (exactitud), la sensibilidad o la especificidad.

6.3 Implementación del algoritmo kNN en R

Para realizar un ejemplo práctico en R (R Core Team, 2023) emplearemos la misma base de datos que empleamos para el modelo *Logit* (Ver Capítulo 3) y para el modelo *Naive Bayes* (Ver Capítulo 5). Recuerden que se trataba de una base de datos provista por Moro et al. (2014)[4]. Recuerda que los datos están relacionados con varias campañas de *marketing* directo de una institución bancaria portuguesa. Las campañas de *marketing* se basaron en llamadas telefónicas. A menudo, se requería más de un contacto con el mismo cliente para determinar si el producto (bank term deposit) sería adquirido ("yes") o no. La base de datos contiene información de 41.188 clientes para 20 variables explicativas y una variable dependiente.

La pregunta de negocio que queremos responder en este caso es: ¿se puede construir un modelo que pueda predecir si un cliente adquirirá o no el producto bancario?

Carga el *working space* que guardaste en el Capítulo 5. Para el detalle del preprocesamiento de la muestra ver el Capítulo 3.

[4]La base de datos está disponible en la página web del libro: http://www.icesi.edu.co/editorial/intro-clasificacion/.

Recordemos que kNN implica el cálculo de distancias entre puntos de datos. Por eso debemos usar solo variables numéricas[5]. La variable de resultado (variable dependiente) debe seguir siendo una variable de clase factor.

Primero, estandaricemos[6] las variables explicativas que deben ser de clase numérico (**numeric**) o entero (**integer**) para evitar que las escalas diferentes tengan un efecto en encontrar los vecinos.

En nuestro caso tendremos las siguientes variables para estandarizar:

- `age`,
- `campaign`,
- `previous`,
- `emp.var.rate`,
- `cons.price.idx`,
- `cons.conf.idx`,
- `euribor3m`.

La estandarización se puede realizar rápidamente con la función **scale()** de la base de R. Para realizar una estandarización, el único argumento que requiere la función es el objeto a estandarizar. Para esto trabajaremos con el objeto `datos` en los que tenemos la base de datos antes de construir las variables dummies y dividir la muestra entre la muestra de entrenamiento y de evaluación. Empleemos el paquete *dplyr*[7] (Wickham et al., 2021) para hacer esta transformación sobre las variables cuantitativas que se emplean como variables independientes.

```
library(dbplyr)
# Estandarizar las variables explicativas
datos_expl <- datos %>%
    dplyr::select(age, campaign, previous, emp_var_rate, cons_price_idx,
↪   cons_conf_idx,
        euribor3m) %>%
    scale()

head(datos_expl, 3)
```

```
##          age   campaign previous emp_var_rate cons_price_idx cons_conf_idx
## 1  1.5330157 -0.5659151 -0.34949    0.6480844      0.7227137     0.8864358
## 2  1.6289735 -0.5659151 -0.34949    0.6480844      0.7227137     0.8864358
## 3 -0.2901821 -0.5659151 -0.34949    0.6480844      0.7227137     0.8864358
##   euribor3m
## 1 0.7124512
## 2 0.7124512
## 3 0.7124512
```

[5]Naturalmente, esto solo aplica para las variables predictoras no para la variable objetivo

[6]Es decir, se le quita la respectiva media para centrar los datos y se divide por la desviación estándar para garantizar que la varianza sea uno.

[7]Ver Alonso (2022) para una introducción al paquete *dplyr*.

6.3.1 Construcción de muestra de entrenamiento y de evaluación

Al igual que lo hicimos en el caso de la estimación de los modelos en los capítulos anteriores, emplearemos el 80% de la muestra para el entrenamiento del algoritmo (la estimación) y el 20% restante para realizar la evaluación. Empleemos los mismos índices para las observaciones que se habían empleado en los capítulos anteriores. Es decir, empleemos los objetos `est_index` y `eval_index` que construimos en el Capítulo 3. Procedamos a crear estas dos muestras.

```
# muestra de estimación
datos_expl_est <- datos_expl[est_index, ]
y_est <- datos$y[est_index]
# muestra de evaluación
datos_expl_eval <- datos_expl[eval_index, ]
y_eval <- datos$y[eval_index]

datos_expl_eval <- as.data.frame(datos_expl_eval)
```

6.3.2 Corriendo el algoritmo kNN

El algoritmo kNN puede implementarse en R con diferente paquetes como *class* (Venables y Ripley, 2002) o *caret* (Kuhn y Max, 2008). Una de las ventajas del segundo paquete es la selección automática del k óptimo. Así, procederemos a emplear el paquete *caret*.

La función **train()** del paquete *caret* permite implementar el algoritmo kNN. Esta función típicamente incluye los siguientes argumentos:

train(x, y, method = "knn", preProcess = NULL)

donde:

- **x**: La matriz de datos que se emplearán para predecir. Es decir, para encontrar los vecinos de i.
- **y**: Un vector que contiene la variable de salida con clase factor.
- **ncp**: k. Número de factores propios.
- **method**: El método para entrenar el algoritmo de inteligencia artificial. Esta función permite implementar varios algoritmos. Para nuestro caso **method = "knn"**.
- **preProcess**: Un vector de texto que especifique el pre-procesamiento que se desee en los datos. Por defecto no se pre-procesan los datos. Si deseamos que los datos sean estandarizados[8], entonces **preProcess = c("center","scale")**. Es importante recordar que la estandarización es para evitar que las escalas de las diferentes variables tengan un efecto en encontrar los vecinos.
- **tuneGrid**: Un objeto de clase **data.frame** que tiene todos los posibles valores de los parámetros que se desean evaluar en una búsqueda de grilla. En nuestro caso, el parámetro será k, el número de vecinos. Este **data.frame** se puede crear con la función **expand.grid()** que establece sobre que valores se va a desarrollar la bús-

[8] Es decir, se le quita la respectiva media para centrar los datos y se divide por la desviación estándar para garantizar que la varianza sea uno.

queda de grilla. En el Capítulo 9 discutiremos en mayor detalle cómo se emplea este argumento para ajustar el modelo.

Noten que nosotros ya estandarizamos las variables cuantitativas por eso no es necesario estandarizar los datos de nuevo. Por otro lado, establezcamos una búsqueda de grilla entre 3 y 10 vecinos.

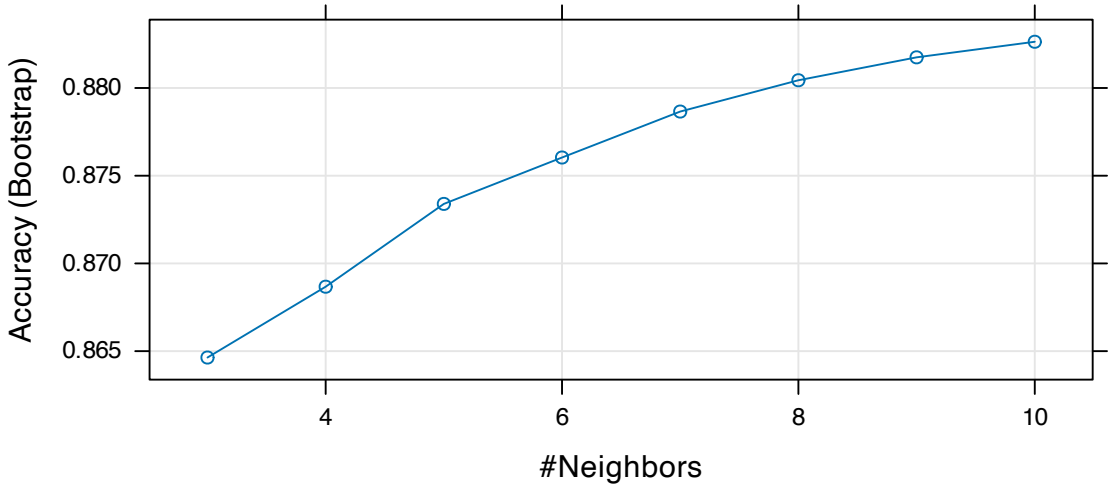
```
library(caret)
set.seed(1234)
kNN_res <- train(datos_expl_est, y_est, method = "knn", preProcess = NULL,
↪  tuneGrid = expand.grid(k = seq(3,
    10, by = 1)))
kNN_res
```

```
## k-Nearest Neighbors
##
## 32950 samples
##     7 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 32950, 32950, 32950, 32950, 32950, 32950, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    3  0.8646359  0.2474794
##    4  0.8686730  0.2513281
##    5  0.8733926  0.2581226
##    6  0.8760372  0.2612837
##    7  0.8786569  0.2641821
##    8  0.8804401  0.2678078
##    9  0.8817470  0.2675983
##   10  0.8826351  0.2683974
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 10.
```

Podemos observar el proceso de selección del k óptimo realizado con la *accuracy* invocando el objeto `kNN_res` y graficando ese mismo objeto con la función **plot()**, los resultados se presentan en la Figura 6.3.

Empleando la precisión, se seleccionó $k = 10$. Ustedes pueden explorar los diferentes compartimientos que posee el objeto `kNN_res` con la función **attributes()**. Recuerde que los resultados en guardados en cada uno de los comportamientos se puede recuperar con el signo `$` antecedido por el nombre del objeto y a continuación el nombre del compartimiento. Por ejemplo,

Figura 6.3. Accuracy del modelo kNN entrenado para diferentes número de vecinos



Fuente: elaboración propia.

```
attributes(kNN_res)
```

```
## $names
##  [1] "method"       "modelInfo"    "modelType"    "results"      "pred"
##  [6] "bestTune"     "call"         "dots"         "metric"       "control"
## [11] "finalModel"   "preProcess"   "trainingData" "ptype"        "resample"
## [16] "resampledCM"  "perfNames"    "maximize"     "yLimits"      "times"
## [21] "levels"
##
## $class
## [1] "train"
```

```
kNN_res$finalModel
```

```
## 10-nearest neighbor model
## Training set outcome distribution:
##
##    no   yes
## 29185  3765
```

Los resultados guardados en esos compartimientos pueden ser útiles en diferentes aplicaciones.

Noten que el número óptimo de vecinos (k) corresponde al límite superior de nuestra búsqueda de grilla. Esto nos lleva a preguntarnos ¿qué pasaría si aumentamos el límite superior de la búsqueda?, ¿seguirá siendo $k = 10$ el valor óptimo? En general siempre deberíamos eligir una grilla lo suficientemente amplia como para estar seguros que el

valor óptimo no se encuentra por fuera de los valores evaluados. En este punto surge una pregunta: ¿cuál es el límite superior de k? En otras palabras, ¿hasta qué valor de k evaluar? Una regla empírica es emplear el máximo k en la búsqueda de grilla como:

$$k = 2 \cdot min\{N, P\} + 1$$

Siguiendo a Dasarathy (1991), una explicación intuitiva de esta regla empírica es la siguiente: considere el caso extremo en el que tenemos un conjunto de datos que contiene P valores positivos ($y = 1$) y solo un valor negativo. En ese caso, si k es mayor a tres, siempre clasificaremos cada individuo como positivo, por lo que la precisión de la validación cruzada será la misma para todos los $k \leq 3$. En ese caso, emplear como número máximo en la búsqueda $k = 3$ sería muy pequeño. Ahora consideremos el otro caso extremo. Supongamos ahora que tenemos 100 observaciones y el mismo número de individuos positivos y negativos $P = N = 50$. En este caso, si fijamos $k = 2N - 1$, entonces clasificar un patrón como positivo o negativo dependerá de la distribución de los datos. Podríamos tener $P = 50$ y $N = 49$ más cercanos a la observación bajo evaluación o $P = 49$ y $N = 50$. Si k es mayor que 50, siempre habrá un empate, ya que sólo tenemos 50 positivos y 50 patrones negativos en la muestra de entrenamiento.

Así, para estar seguros de que hemos analizado todos los valores no redundantes de k es posible que tengamos que buscar hasta el número total de muestras de entrenamiento. Por ejemplo, si tenemos $N < P$, entonces tendríamos que buscar hasta $k = 2N + 1$ (ya que un **kNN** con k mayor a $2N + 1$ implicará más observaciones votadas como positivas que negativas). Y algo similar ocurrirá cuando $N > P$.

En nuestro caso $P = 3765$ y $N = 29185$, así el k máximo para nuestra búsqueda de grilla sería $2 \cdot min\{N, P\} + 1$ = 7531. Esto tomará mucho tiempo de computo y memoria. Tu puedes comprobar que no es necesaria una búsqueda tan grande. Para agilizar el proceso de cómputo solo evaluaremos hasta un valor de k de 150. Así tendremos:

```
# Establecer la grilla de búsqueda para el costo


kNN_res <- train(datos_expl_est, y_est, method = "knn", preProcess = NULL,
↪  tuneGrid = expand.grid(k = seq(9,
    150, by = 1)))

kNN_res$finalModel
```

```
## 111-nearest neighbor model
## Training set outcome distribution:
##
##    no   yes
## 29185  3765
```

```
# No se muestra esta visualización intenta construir esta visualización
# plot(kNN_res)
```

Empleando la precisión, se seleccionó $k = 111$. Noten que en ese caso el óptimo no fue cerca al límite superior de la grilla de búsqueda.

6.3.3 Bondad de ajuste del modelo

Ahora generemos la matriz de confusión y las correspondientes métricas. Primero debemos generar las predicciones fuera de muestra para la clase. Esto se puede hacer de manera similar a lo que hicimos en el caso del modelo*Logit* y para el modelo *Naive Bayes*.

```
pred_kNN <- predict(kNN_res, newdata = datos_expl_eval, type = "raw")
```

Y ahora veamos la correspondiente matriz de confusión:

```
confusionMatrix(pred_kNN, y_eval, positive = "yes")$table
```

```
##            Reference
## Prediction   no  yes
##        no  7244  729
##        yes  119  146
```

Cuadro 6.1. Métricas de comparación de modelos para modelo Logit, Naive Bayes y kNN

	Accuracy	Sensitivity	Specificity	Precision	F1
Modelo Logit	0.846	0.618	0.873	0.366	0.460
Naive Bayes	0.854	0.472	0.899	0.357	0.407
kNN	0.897	0.167	0.984	0.551	0.256

Fuente: elaboración propia.

Las otras métricas estudiadas del mejor modelo Logit, el *Naive Bayes* y kNN se reportan en el Cuadro 6.1.

Noten que el modelo kNN tiene el mejor *accuracy* (89.7 %), sin embargo, clasificando el valor positivo "yes", no es muy bueno. Además este modelo si bien tiene una proporción alta de falsos positivos, esta no es tan alta como los otros dos modelos. El modelo kNN clasifica a 119[9] observaciones en la clase "yes", que realmente no fueron "yes". Para el caso del modelo *Naive Bayes* el porcentaje de falsos positivos era de 64.3 % y para el modelo *Logit* esta tasa de falsos positivos era de 63.4 %.

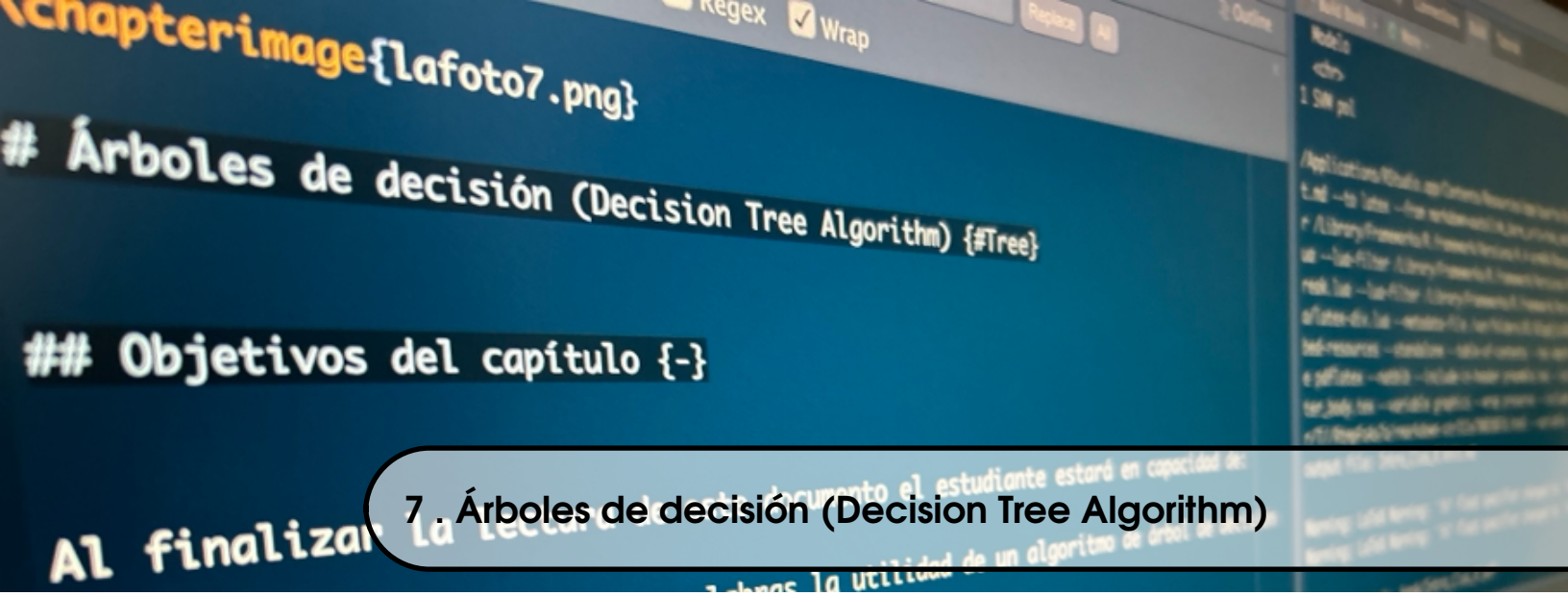
Finalmente, guarda el espacio de trabajo para ser empleado en los siguientes capítulos.

[9]Esto representa el 44.9 % de todos los positivos predichos por el modelo *Naive Bayes*.

6.4 Comentarios Finales

En este capítulo hemos discutido el método de aprendizaje supervisado comúnmente empleado en las tareas de clasificación. En los próximos capítulos estudiaremos otros métodos de aprendizaje de máquinas que también sirven para clasificar. Como se mencionó anteriormente, *a priori*, es imposible saber si este método será mejor que el estadístico u otros de aprendizaje de máquina y por eso es común que se empleen diferentes aproximaciones y se comparen.

Las aplicaciones de estos métodos de clasificación son muchas en el mundo de los negocios y definitivamente estos modelos son necesarios en la caja de herramientas de un científico de datos.



7 . Árboles de decisión (Decision Tree Algorithm)

Objetivos del capítulo

Al finalizar la lectura de este documento el estudiante estará en capacidad de:

- Explicar en sus propias palabras la utilidad de un algoritmo de árbol de decisión (*Decision Tree Algorithm*).
- Emplear un algoritmo de clasificación de árbol en R.
- Calcular métricas de la bondad de un modelo de clasificación estimado por medio de un algoritmo de clasificación de árbol.
- Encontrar los pronósticos de clasificación de un algoritmo de clasificación de árbol.

7.1 Introducción

Anteriormente, discutimos que los modelos de clasificación que se emplean en el **business analytics** tienen dos orígenes: modelos estadísticos y modelos de aprendizaje de máquinas[1]. En los anteriores capítulos discutimos el modelo estadístico *Logit* (el más empleado para clasificación) y el modelo *Naive Bayes*. Posteriormente, estudiamos una técnica de aprendizaje de máquina conocida como kNN. En esta oportunidad nos concentraremos en otra técnica de inteligencia artificial conocida como árboles de decisión (*Decision Trees*).

Los árboles de decisión son un método de aprendizaje supervisado no paramétrico que puede ser empleado para hacer tareas de clasificación y de regresión. Esta característica hace único a este algoritmo, pues es de los pocos que permiten hacer cualquiera de estas dos tareas. En general, los árboles de decisión aprenden de los datos para crear un conjunto de reglas de decisión del tipo si-entonces-otro (*if-then-else*);

[1] También conocido como aprendizaje automático o aprendizaje automatizado. Este término viene del inglés *machine learning*.

reglas que están anidadas. Este algoritmo no calcula parámetros (como el modelo *Logit*) y por eso se clasifica como un método no paramétrico.

Si la variable de salida es una variable categórica (Árboles de decisión con variable categórica **Categorical Variable Decision Tree**), entonces la tarea que resuelven los árboles de decisión son de clasificación. Si la variable de salida es una variable continua, entonces los árboles de decisión resuelven tareas de regresión.

En cualquiera de los dos casos, la idea detrás del algoritmo es relativamente sencilla. La filosofía de los árboles de decisión se basa en la máxima "divide y vencerás". Es decir, la idea es dividir la muestra cada vez en dos grupos, tal que las observaciones al interior de los grupos se parezcan lo más posible. En esta literatura a esto se le conoce como ser lo más "puro posible". Es decir, cuando los subgrupos tienen clasificados individuos que solo pertenecen a una clase, los llamaremos puros; en caso contrario los llamaremos impuros. Veamos un ejemplo para entender este concepto.

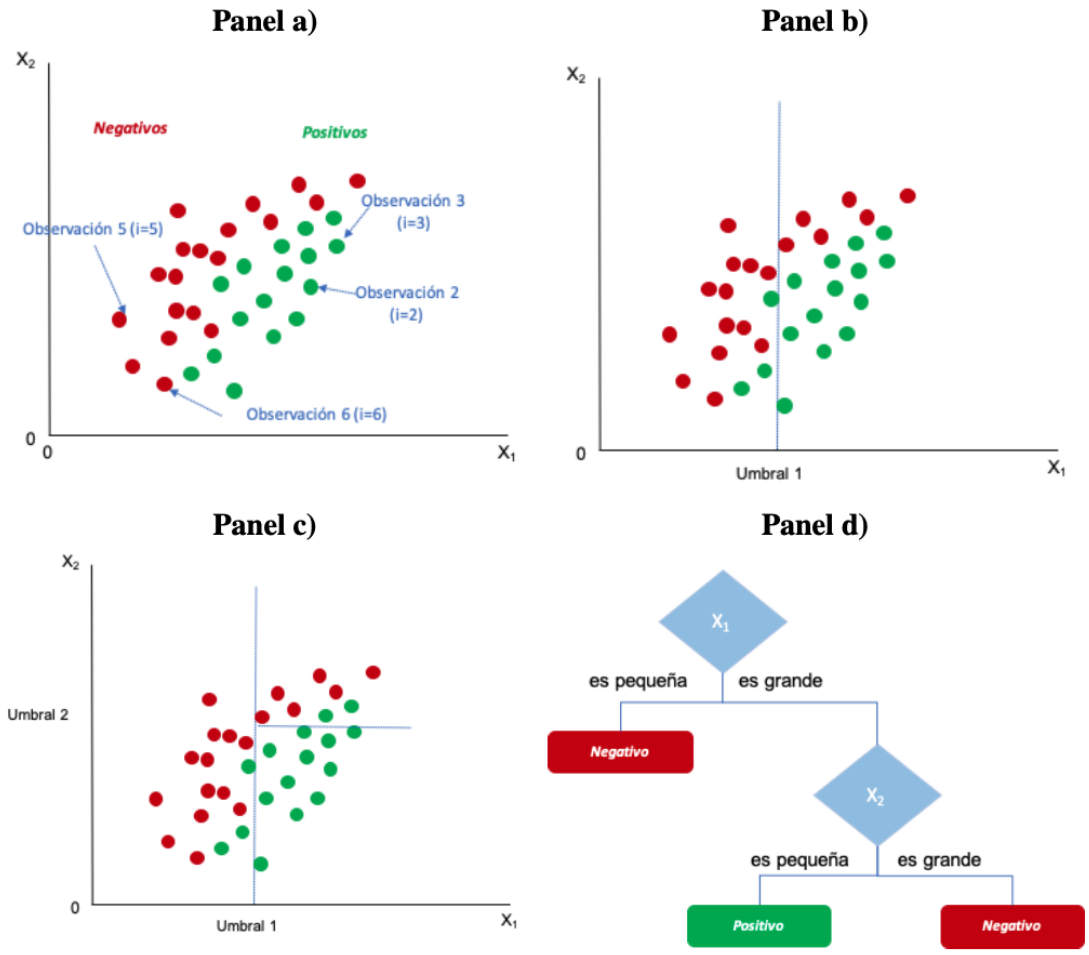
Por ejemplo, supongamos que contamos con dos clases: compra el producto (positivo) y no lo compra (negativo) y tenemos dos variables: nivel de ingresos ($X_{1,i}$) y los años de educación ($X_{2,i}$). (Ver Figura 7.1 panel a)).

Ahora, el algoritmo "escoge" una variable para dividir la muestra. Supongamos que empezamos empleando el ingreso (X_1) como la variable que mejor divide en el primer paso la muestra en los que compran el producto (positivo) y los que no lo compran (negativo). El algoritmo establece un umbral (Umbral 1) que permite dividir el espacio de la variable ingresos (X_1) en dos regiones los ingresos bajos y los altos (Ver Figura 7.1 panel b)).

El segundo paso es dividir las dos áreas empleando las siguientes variables en este caso los años de educación (X_2). En este caso en especial es claro para el algoritmo, que para valores pequeños de X_1 no es necesario dividir de nuevo la muestra, pues prácticamente casi todos los valores son negativos. Pero para valores grandes de X_1 el algoritmo puede encontrar un segundo umbral (Umbral 2), pero esta vez para la variable años de educación (X_2). Nuevamente, la muestra queda dividida en dos grupos: valores bajos de X_2 (por encima del Umbral 2) y valores altos (por debajo del Umbral 2) (ver Figura 7.1 panel b)). Noten que el algoritmo encontró dos reglas del tipo si-entonces-otro (*if-then-else*). La primera regla es si el ingreso ($X_{1,i}$) es menor que el umbral 1, entonces i será un negativo (no compra el producto). En caso contrario tenemos una segunda regla para determinar la clasificación de i. La segunda regla implica determinar si los años de educación del individuo i ($X_{2,i}$) es menor que el umbral 2, entonces i se clasificará como un positivo (se comprará el producto). En caso de que $X_{2,i}$ sea mayor que el umbral 2, i se etiquetará como un negativo (no compra el producto).

Estas reglas que acaba de encontrar el algoritmo se pueden representar en forma de un árbol de decisión como el que se presenta en el panel d) de la Figura 7.1. Los puntos de decisión (representados por rombos en este caso) se conocen como nodos de decisión. Los nodos de decisión implican tener un umbral (una regla de decisión) que divide a la muestra (el proceso de dividir la muestra se denomina en inglés **split**). El primer nodo (el que se encuentra más arriba) se denomina nodo raíz (ver Figura

Figura 7.1. Funcionamiento del algoritmo de clasificación de árbol para variables categóricas



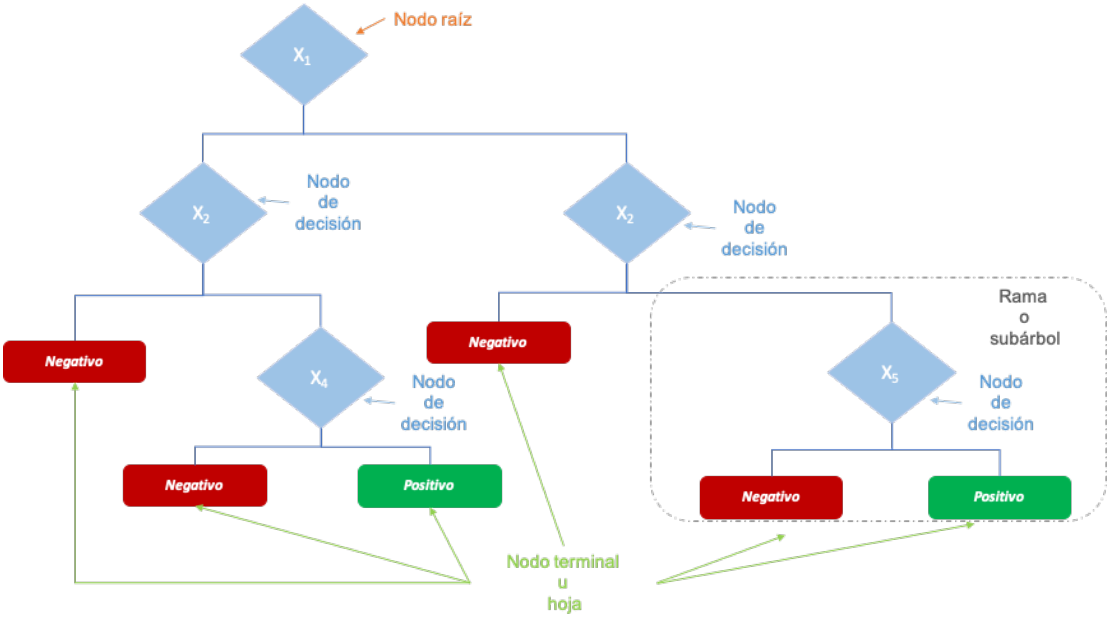
Fuente: elaboración propia.

7.2). Los nodos terminales u hojas corresponden a los nodos que no dividen más a la muestra. Estos nodos tienen flechas que entran en ellos, pero ninguna flecha sale de ellos.

Antes de continuar, es importante reconocer que los tres nodos terminales u hojas que se formaron con la muestra, dos son impuros; mientras que uno es puro. Es decir, al aplicar la primera regla obtenemos un nodo terminal en el que "caen" 16 de los individuos que serán etiquetados como negativos; los que tienen ingresos por debajo del umbral 1. En este nodo terminal, 3 son positivos, los otros 13 fueron reales negativos. Por eso se considera como una hoja impura o, dicho de otra forma, no todos los individuos pertenecen a una clase. Por otro lado, al aplicar la segunda regla a los individuos (i) con ingresos mayores al umbral 1, encontramos un nodo terminal de 11 individuos con educación menor al umbral 2. Este nodo terminal implica que los individuos son clasificados como positivos. Los 11 individuos de esta hoja son clasificados correctamente como positivos. Es decir, el nodo terminal es totalmente puro. La tercera hoja tiene 2 individuos de los 8 que son clasificados erróneamente como negativos. Por eso es un nodo terminal impuro.

En la Figura 7.2 se presenta un árbol de decisión un poco más complicado. Una Rama (*Branch*) o Subárbol es una subsección del árbol. El proceso de eliminar ramas se denomina poda (*pruning*). Crear más ramas se conoce como *splitting*. Finalmente, un nodo de decisión que se divide en subnodos, se denomina nodo padre, mientras que los subnodos se conocen como nodos hijos.

Figura 7.2. Terminología de los árboles de clasificación



Fuente: elaboración propia.

7.2 Podando árboles

Es importante anotar que uno de los problemas de este tipo de algoritmo es que los árboles de decisión tienden a convertirse en bastante complejos rápidamente (grandes). Lo cuál esta asociado con el *overfitting*. Es decir, que el modelo quede demasiado entrenado para la muestra de tal manera que no sea posible actuar bien en otra muestra (nuevos datos que quisiéramos clasificar en el futuro). Para evitar este crecimiento se usan técnicas de "poda" (*pruning*).

Un método es el que se conoce como pre-poda (*pre-pruning*). Esto implica parar el "crecimiento del árbol cuando este alcanza un determinado tamaño. Por ejemplo, se le puede pedir al algoritmo que pare cuando ha llegado un determinado nivel de profundidad (Ver Figura 7.3 panel a). Otra técnica de pre-poda (Ver Figura 7.3 panel b) implica parar el crecimiento de una rama cuando se alcance un mínimo de observaciones.

Parar el "crecimiento" de un árbol en un momento temprano, puede evitar que este encuentre patrones importantes. Para tratar de evitar esto pueden emplearse técnicas de post-poda. Por ejemplo (Ver Figura 7.3 paneles c) y d)), se puede dejar crecer un árbol tan grande como se pueda y después podar algunas ramas usando la contribución a minimizar errores de cada rama.

Miremos esto de una manera intuitiva; un árbol se puede entender como un gran cuestionario que se le aplica a los datos, cada rama representa preguntas. Ese cuestionario grande nos lleva a una respuesta final: la clasificación de cada individuo (hoja o nodo terminal). Cada pregunta divide el grupo de datos en ramas más pequeñas, acercándonos a la clase a la que pertenece cada individuo.

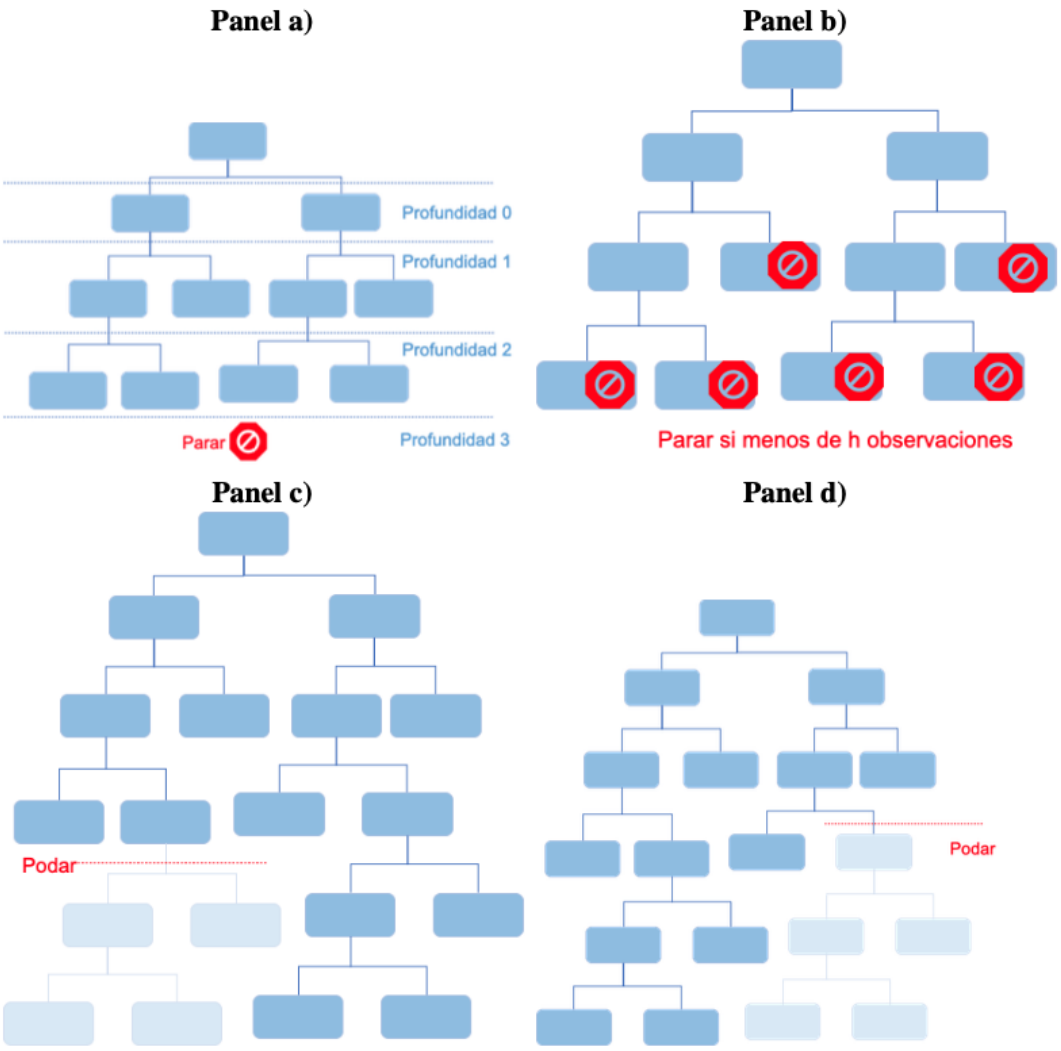
La complejidad de un árbol de clasificación tiene que ver con su tamaño y profundidad (número de ramas). Un árbol con pocos niveles y ramas es simple, mientras que uno con muchos niveles y ramas intrincadas es más complejo.

Un árbol muy simple podría no capturar todas las relaciones importantes en los datos, lo que conduce a errores en las clasificaciones, lo que llevaría a falta de precisión en la clasificación; pero un árbol con muchos niveles nos puede llevar al *overfitting*[2]. Adicionalmente, un árbol complejo puede ser difícil de entender y seguir, haciendo que sea poco práctico. Y en algunos casos, los árboles complejos requieren más memoria RAM para almacenar y más tiempo para procesar, lo cual puede ser inconveniente si se cuenta con una base de datos muy grande. Por eso es importante "podar" un árbol. Es decir, eliminar ramas que no aportan mucho a la clasificación, buscando un equilibrio entre precisión y complejidad.

Como vimos, podar es cortar ramas innecesarias para mantener el árbol lo más pequeño y preciso posible. Como se discutió arriba, las técnicas de poda son variadas. Por ejemplo, se puede definir un límite de profundidad o número de ramas máximo para el árbol; a esta técnica se le llama pre-poda. Es decir, se para el crecimiento antes que necesitemos podar el árbol. Para la pre-poda se pueden emplear diferentes criterios, como el número de niveles o el número de observaciones por rama. Por otro

[2] Ver la Sección 2.2 para una discusión de este término.

Figura 7.3. Técnicas de poda



Fuente: elaboración propia.

lado, está la posibilidad de que el árbol crezca y se corten ramas innecesarias: post-poda.

En últimas, queremos aplicarle un "cuestionario" a los individuos que nos permita realizar la clasificación sin que éste sea muy complejo.

En la post-poda se emplea a menudo un parámetro denominado el **parámetro de complejidad** (cp). El cp es la mejora mínima del modelo necesaria en cada nodo para permitir que ese nodo no sea cortado. El cp intenta resumir en un número el costo de la complejidad del árbol[3] sumando la tasa de casos mal clasificados (en los nodos terminales) ($R(T)$) y el número de divisiones (*splits*) (T) multiplicado por un término de penalización α ($\alpha > 0$). Es decir,

$$cp(T, \alpha) = R(T) + \alpha T$$

Así, el **parámetro de complejidad** es función del número de *splits* (T) del árbol y del término de penalización que se convierte en un parámetro. En la práctica, se puede emplear el cp para acelerar la búsqueda de divisiones porque puede identificar divisiones que no cumplen este criterio y eliminarlas antes de ir demasiado lejos.

7.3 Selección de umbrales

Antes de pasar a estudiar cómo hacer esto en R, la pregunta que queda es ¿cómo hace el algoritmo para seleccionar los umbrales? Como se mencionó anteriormente, la idea del método de árboles de decisión es dividir la muestra de tal manera que se creen grupos homogéneos. Es decir, que se creen nodos terminales puros.

Esto implica, que el proceso de separación del algoritmo necesitará emplear algún tipo de medida de no pureza (heterogeneidad). Existen diferentes medidas de impuridad como por ejemplo el Coeficiente de Gini, Coeficiente de Entropía, la Ganancia de información, la Razón de ganancia, la Reducción de varianza y Chi-cuadrado. Concentrémonos en los dos primeros.

El coeficiente de Gini[4] en este contexto para un problema de clasificación binario para cada uno de los dos posibles nodo hijo (j) se define como:

$$Gini_j = 1 - (p_j)^2 - (q_j)^2 \tag{7.1}$$

donde p_j es la proporción de positivos en el nodo hijo j y q_j es la proporción negativos en el mismo nodo. Entonces, para un nodo padre al encontrar un umbral, se calculan el coeficiente de Gini para los dos nodos hijos y se crea un promedio ponderado por los casos (individuos) que son clasificados en cada uno de los nodos hijos como proporción de los casos que llegan al nodo padre. Es importante recalcar que entre

[3] Cuando se emplea el cp para hacer la poda, a la aproximación se le conoce como *Cost-complexity pruning* (CCP).

[4] Los economistas usan el coeficiente de Gini como una medida de desigualdad en la distribución del ingreso.

más pequeño sea el coeficiente de Gini para el nodo j , mejor la tarea de separar los individuos (mayor será la homogeneidad y por tanto su pureza).

El algoritmo de clasificación con un árbol de decisión empleando como medida de impureza el coeficiente de Gini implicará los siguientes pasos:

1. Encontrar el umbral para cada variable continua que minimice el coeficiente de Gini ponderado (la suma ponderada de los dos coeficientes de Gini de los nodos hijos). Es decir, encontrar el valor de la variable continua que genere el mejor "split" de la muestra. Esto implica evaluar todos los posibles umbrales (punto medios entre todas las observaciones) para cada variable.
2. Calcular los coeficientes de Gini para los nodos de las variables dicotómicas. En caso de existir variables categóricas con mas de una clase, se agrupan las clases de tal manera que se minimice el coeficiente de Gini para esa variable.
3. Seleccionar la variable con el coeficiente de Gini más bajo como la variable que corresponderá al nodo raíz, y generar los dos nuevos nodos.
4. Repetir 1 a 2 para cada nuevo nodo.
5. Seleccionar la variable con el coeficiente de Gini ponderado mejor como el nuevo nodo de decisión.
6. Repetir 4 y 5 hasta que se encuentre un nodo para el cual no existe más variables que presten un coeficiente de Gini que sea menor al coeficiente de Gini del nodo padre o hasta que no existan más variables a evaluar. En este caso se encontró un nodo terminal.

A este algoritmo se le conoce como CART (por su sigla en inglés *Classification and Regression Tree*[5].) .

Otra medida de desigualdad muy empleada es el Coeficiente de Entropía también conocido como índice de información (*information index*). Este coeficiente es una medida de la aleatoriedad en la información que se procesa. Cuanto mayor es la entropía, más difícil es sacar conclusiones de esa información. En este contexto, para un problema de clasificación binario para cada uno de los dos posibles nodos hijos (j) se define como:

$$E_j = 1 - p_j log(p_j) - q_j log(q_j) \tag{7.2}$$

Al igual que en el caso del coeficiente de Gini, para cada posible nodo de decisión se calcula el promedio ponderado de cada uno de los dos nodos hijos.

En este caso, el algoritmo que emplea el criterio de entropía para armar el árbol de decisión se conoce como ID3. Este algoritmo sigue una regla sencilla: un nodo con una entropía de cero será un nodo terminal (hoja) y un nodo con una entropía mayor que cero necesita una división adicional.

A priori, es imposible determinar qué algoritmo (con el índice de Gini o el de entropía u otro) para construir árboles de decisión es mejor, por eso una buena práctica es emplear varios criterios y evaluar su comportamiento. Tampoco es posible saber *a priori* si el árbol de decisión hace mejor tarea o no que un modelo *Logit*, *Naive Bayes* o que

[5]Este algoritmo se le atribuye a Breiman et al. (1984)

el algoritmo kNN. Por eso es común calcular todas estas opciones y compararlos con las métricas que ya conocemos.

En las siguientes secciones veremos cómo correr este algoritmo en R y comparar los resultados con otros métodos de clasificación.

7.4 Implementación del algoritmo en R

Para realizar un ejemplo práctico en R (R Core Team, 2023) emplearemos la misma base de datos que empleamos para el modelo *Logit* (Capítulo 3), el modelo *Naive Bayes* (Capítulo 5) y el algoritmo kNN (Capítulo 6).

Recuerda que la pregunta de negocio que queremos responder en este caso es: ¿se puede construir un modelo que pueda predecir si un cliente adquirirá o no el producto bancario?

Carga el *working space* que guardaste en el Capítulo 6. Para el detalle del preprocesamiento de la muestra ver el Capítulo 3.

7.4.1 Corriendo el algoritmo con el criterio de Gini

El algoritmo de construcción de árboles de clasificación, tanto empleando el coeficiente de Gini como el de entropía, se puede implementar en R con diferentes paquetes. En esta ocasión emplearemos el paquete *rpart* (Therneau y Atkinson, 2022).

La función **rpart()** del paquete *rpart* permite implementar este algoritmo y muchos otros de partición. En nuestro caso solo necesitamos los siguientes argumentos de esta función:

rpart(formula, data, method, parms, control)

donde:

- **formula**: Una fórmula que determina la variable dependiente (de salida) y las variables de entrada (las que irán en los nodos).
- **data**: Un objeto de clase **data.frame** que contiene los datos.
- **method**: para nuestro caso tenemos un método de clasificación, por eso usaremos **method = "class"**.
- **parms**: argumento que permite definir el criterio para generar los *splits*. Si **parms = list(split = "information")** se emplea el coeficiente de entropía (o índice de información). Si **parms = list(split = "gini")** se emplea el coeficiente de Gini. Este último es el criterio por defecto.
- **control**: Este argumento permite controlar el crecimiento del árbol.

Empecemos creando un árbol con el criterio de Gini para separar las muestras:

```
set.seed(123)
library(rpart)
```

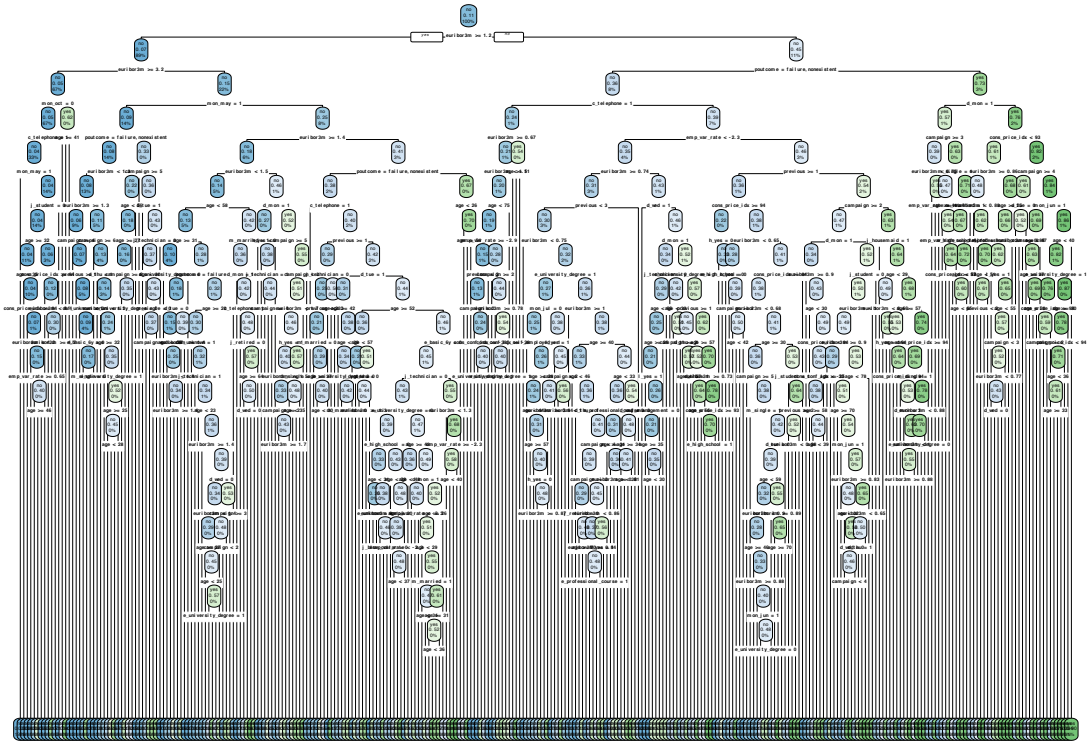
```
RT_res_gini <- rpart(formula = y ~ ., data = datos_dummies_est, method =
↪  "class",
    parms = list(split = "gini"), control = rpart.control(cp = 0))
```

Podemos observar el árbol de decisión empleando la función **rpart.plot()** del paquete *rpart.plot* (Milborrow, 2022).

```
library(rpart.plot)
rpart.plot(RT_res_gini)
```

Este árbol es bastante complejo (Ver Figura 7.4). Necesitamos hacer una poda.

Figura 7.4. Árbol sin podas



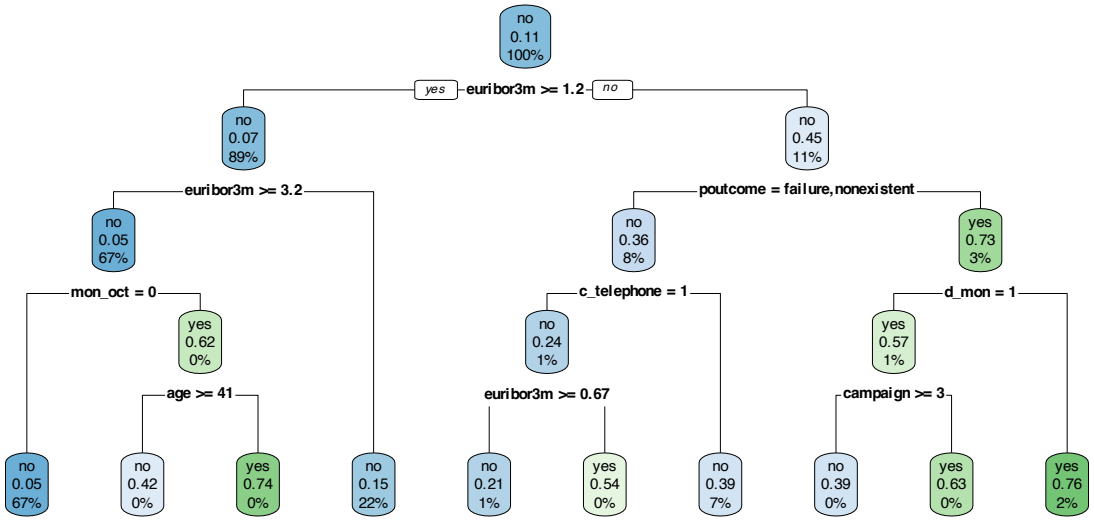
Fuente: elaboración propia.

7.4.1.1 Poda

Empleemos las técnicas de pre-poda discutidas anteriormente. Por ejemplo, limitemos el crecimiento del árbol hasta 4 niveles (Ver Figura 7.6):

```
RT_res_gini_4niveles <- rpart(y ~ ., data = datos_dummies_est, method =
↪  "class",
    parms = list(split = "gini"), control = rpart.control(cp = 0, maxdepth =
    ↪  4))
```

Figura 7.5. Árbol podado hasta 4 niveles



Fuente: elaboración propia.

El árbol con 4 niveles se puede observar en la Figura 7.5. Ahora intentemos para el crecimiento cuando los nodos no tengan mas de 500 observaciones.

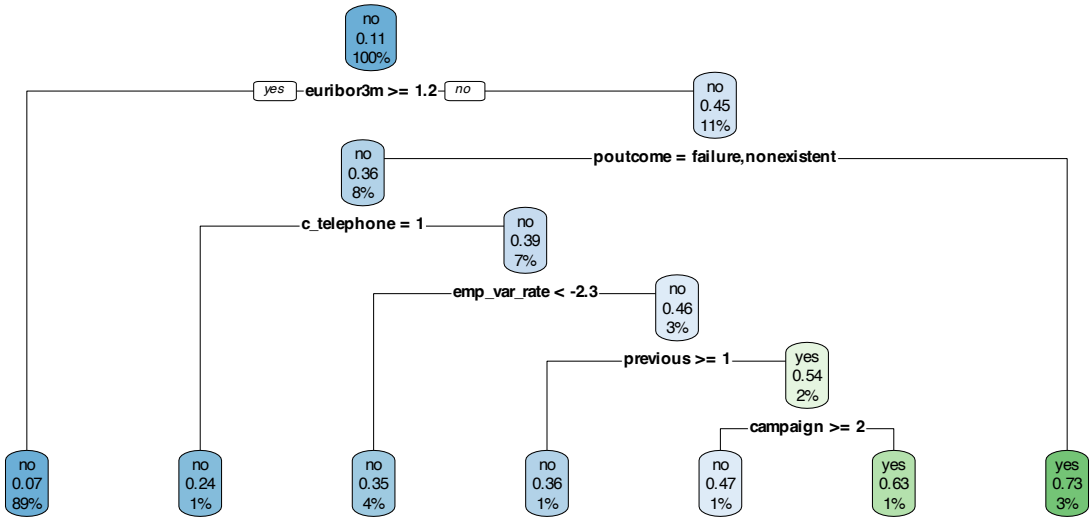
```
RT_res_gini_500obs <- rpart(y ~ ., data = datos_dummies_est, method = "class",
↪  parms = list(split = "gini"),
    control = rpart.control(cp = 0, minsplit = 500))
```

Ahora consideremos la posibilidad de una post-poda partiendo del árbol de decisión que construimos inicialmente (el objeto `RT_res_gini`). Como se explicó anteriormente, para establecer dónde realizar la post-poda se puede emplear el parámetro de complejidad (cp) del árbol. Estamos buscando el cp que minimice el error del modelo (columna **xerror** en la próxima salida de R). El cp para los diferentes *splits* y sus respectivos errores se pueden extraer del objeto de clase **rpart** empleando la función **printcp()**.

```
printcp(RT_res_gini)
```

```
##
## Classification tree:
## rpart(formula = y ~ ., data = datos_dummies_est, method = "class",
##     parms = list(split = "gini"), control = rpart.control(cp = 0))
##
## Variables actually used in tree construction:
##  [1] age                 c_telephone         campaign
##  [4] cons_conf_idx       cons_price_idx      d_mon
```

Figura 7.6. Árbol con ramas con 500 observaciones o más



Fuente: elaboración propia.

```
##  [7] d_thu                 d_tue                 d_wed
## [10] def_unknown           e_basic_6y            e_high_school
## [13] e_professional_course e_university_degree   e_unknown
## [16] emp_var_rate          euribor3m             h_yes
## [19] j_blue_collar         j_housemaid           j_management
## [22] j_retired             j_self_employed       j_student
## [25] j_technician          l_yes                 m_married
## [28] m_single              mon_jul               mon_jun
## [31] mon_may               mon_oct               poutcome
## [34] previous
##
## Root node error: 3765/32950 = 0.11426
##
## n= 32950
##
##           CP nsplit rel error  xerror     xstd
## 1  5.1129e-02      0   1.00000 1.00000 0.015338
## 2  3.1873e-03      2   0.89774 0.89774 0.014628
## 3  2.3904e-03      6   0.88313 0.89270 0.014592
## 4  2.1780e-03      7   0.88074 0.88818 0.014559
## 5  1.3280e-03     12   0.86985 0.88499 0.014536
## 6  1.1952e-03     18   0.86189 0.88712 0.014551
## 7  1.1510e-03     20   0.85950 0.89296 0.014594
## 8  1.0624e-03     23   0.85604 0.89270 0.014592
## 9  9.2961e-04     37   0.84117 0.89456 0.014605
```

```
## 10 8.8535e-04     45   0.83267 0.89323 0.014596
## 11 7.9681e-04     48   0.83001 0.89588 0.014615
## 12 7.3779e-04     63   0.81753 0.90120 0.014653
## 13 6.6401e-04     72   0.81089 0.90305 0.014666
## 14 6.1974e-04     77   0.80744 0.90518 0.014682
## 15 5.7548e-04     80   0.80558 0.90518 0.014682
## 16 5.3121e-04     88   0.80027 0.90571 0.014686
## 17 4.7809e-04    102   0.79283 0.91049 0.014720
## 18 4.6481e-04    108   0.78991 0.91235 0.014733
## 19 4.4267e-04    112   0.78805 0.91262 0.014735
## 20 3.9841e-04    127   0.78088 0.91501 0.014752
## 21 3.5414e-04    148   0.77185 0.91952 0.014784
## 22 3.3201e-04    151   0.77078 0.92058 0.014792
## 23 3.2463e-04    155   0.76946 0.92191 0.014801
## 24 2.6560e-04    167   0.76388 0.94130 0.014937
## 25 1.9920e-04    230   0.74422 0.94210 0.014943
## 26 1.3280e-04    238   0.74263 0.94475 0.014961
## 27 1.2499e-04    258   0.73997 0.94927 0.014993
## 28 1.1383e-04    276   0.73758 0.94954 0.014995
## 29 8.8535e-05    283   0.73679 0.95139 0.015007
## 30 6.6401e-05    292   0.73599 0.95325 0.015020
## 31 5.3121e-05    296   0.73572 0.95644 0.015042
## 32 0.0000e+00    301   0.73546 0.96600 0.015108
```

En este caso el error mínimo (*xerror*) se encuentra para 12 *splits* y corresponde a un cp de 0.001328. Esto se puede ver gráficamente empleando la función **plotcp()**, como se presenta en la Figura 7.7.

```
plotcp(RT_res_gini)
```

En la Figura 7.7 se presenta el parámetro de complejidad (cp) (eje horizontal, parte inferior) del árbol y el error relativo (eje vertical). Ahora, podemos el árbol empleando un cp de 0.001328. El siguiente código produce el árbol podado que se presenta en la Figura 7.8.

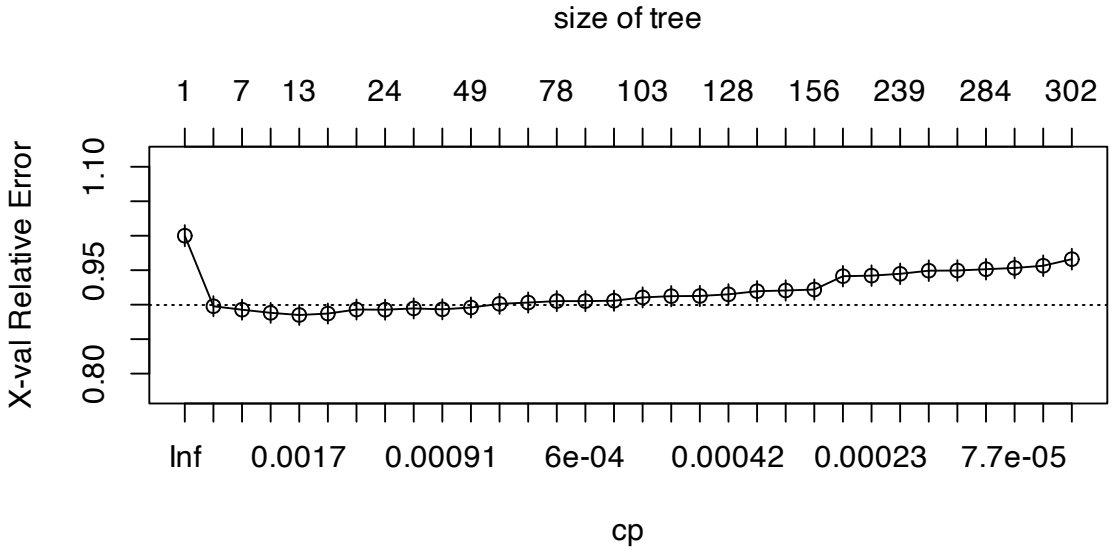
```
cp_optimo <- res[which.min(res[, 4]), 1]
RT_res_gini_podado <- prune(RT_res_gini, cp = cp_optimo)
```

7.4.2 Corriendo el algoritmo con el criterio de entropía

Ahora podemos repetir el ejercicio empleando el índice de entropía como criterio para la separación de las muestras. Es decir, primero corramos el árbol sin ninguna poda con el siguiente código:

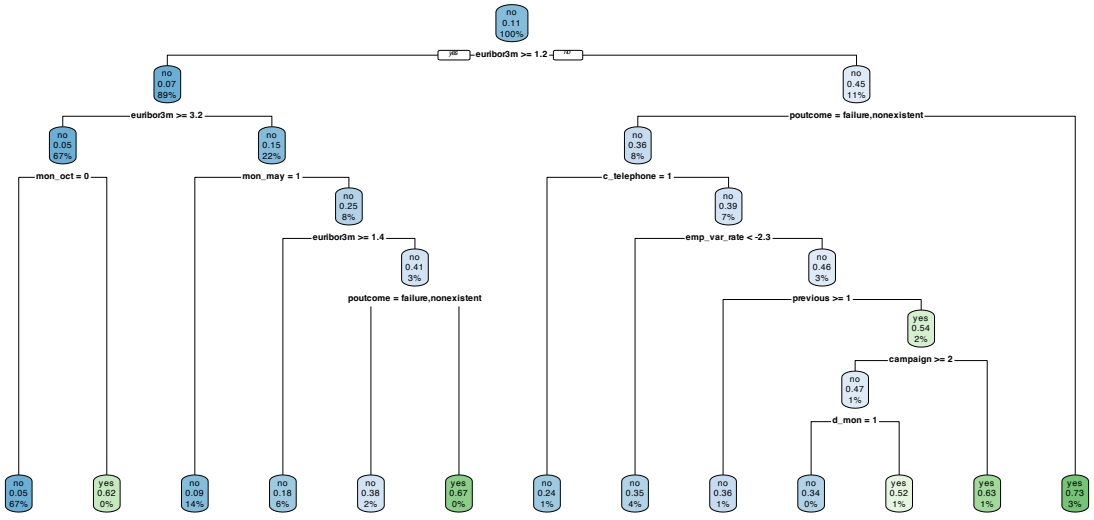
```
RT_res_info <- rpart(formula = y ~ ., data = datos_dummies_est, method =
↪  "class",
    parms = list(split = "information"), control = rpart.control(cp = 0))
```

Figura 7.7. Parámetro de complejidad (cp) y error relativo (xerror) para el modelo del árbol construido con el criterio de Gini



Fuente: elaboración propia.

Figura 7.8. Árbol construido con el criterio de Gini post-podado empleando el parámetro de complejidad



Fuente: elaboración propia.

Ahora empleemos como método de pre-poda una profundidad máxima de 4 niveles, y guardemos los resultados en el objeto `RT_res_info_4niveles`.

```
RT_res_info_4niveles <- rpart(formula = y ~ ., data = datos_dummies_est, method
↪  = "class",
    parms = list(split = "information"), control = rpart.control(cp = 0,
    ↪  maxdepth = 4))
```

Y ahora empleemos como criterio de pre-poda que el número máximo de observaciones por rama sea de 500. Esto lo podemos hacer con el siguiente código:

```
RT_res_info_500obs <- rpart(formula = y ~ ., data = datos_dummies_est, method =
↪  "class",
    parms = list(split = "information"), control = rpart.control(cp = 0,
    ↪  minsplit = 500))
```

Finalmente, empleemos una post-poda con el parámetro de complejidad (cp).

```
printcp(RT_res_info)
```

```
# Guardar los resultados del cp en un objeto para encontrar el mínimo con una
# función
res2 <- printcp(RT_res_info)
```

```
# Encontrando el cp que minimiza el xerror (4 columna)
res2[which.min(res2[, 4]), 1]
```

```
## [1] 0.001328021
```

```
cp_optimo <- res2[which.min(res2[, 4]), 1]
```

```
# Podando el árbol con el cp óptimo
RT_res_info_podado <- prune(RT_res_info, cp = cp_optimo)
```

7.4.3 Bondad de ajuste del modelo

Ahora comparemos el ajuste de los seis modelos podados: tres podados con el criterio de Gini y tres con el de entropía.

Ahora generemos la matriz de confusión y las correspondientes métricas, para determinar cuál de todas las seis aproximaciones es la mejor. Primero debemos generar las predicciones fuera de muestra para la clase. Esto se puede hacer de manera similar a lo que hicimos en el caso del modelo *Logit*, *Naive Bayes* y el kNN.

```
pred_RT_res_gini_4niveles <- predict(RT_res_gini_4niveles, newdata =
↪  datos_dummies_eval,
    type = "class")
```

```
mo_gini_4niveles
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  7305  720
##        yes   58  155
##
##                Accuracy : 0.9056
##                  95% CI : (0.899, 0.9118)
##     No Information Rate : 0.8938
##     P-Value [Acc > NIR] : 0.0002286
##
##                   Kappa : 0.2539
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.17714
##             Specificity : 0.99212
##          Pos Pred Value : 0.72770
##          Neg Pred Value : 0.91028
##              Prevalence : 0.10622
##          Detection Rate : 0.01882
##    Detection Prevalence : 0.02586
##       Balanced Accuracy : 0.58463
##
##        'Positive' Class : yes
##
```

Ahora replica esto para los otros 5 modelos

En el Cuadro 7.1 se reportan las métricas que permiten comparar todos los modelos que hemos estimado hasta ahora.

Hasta aquí el mejor modelo en términos de *accuracy* es el que hemos denominado RT_gini_podado. En términos de sensibilidad el mejor modelo es Modelo Logit. En especificidad el mejor modelo es el RT_gini_4niveles. En precisión el mejor modelo es RT_gini_4niveles. Y en términos del puntaje F_1, el mejor modelo es Modelo Logit. Decidir entre estos modelos no es una tarea fácil y tendrá que ser el negocio, en especial el *analytics translator* con los tomadores de decisiones, que tendrán que determinar cuál métrica es más conveniente para el negocio y el uso que se le dará al modelo seleccionado.

Finalmente, guarda el espacio de trabajo para ser empleado en los siguientes capítulos.

Cuadro 7.1. Métricas de comparación para todos los modelos de clasificación

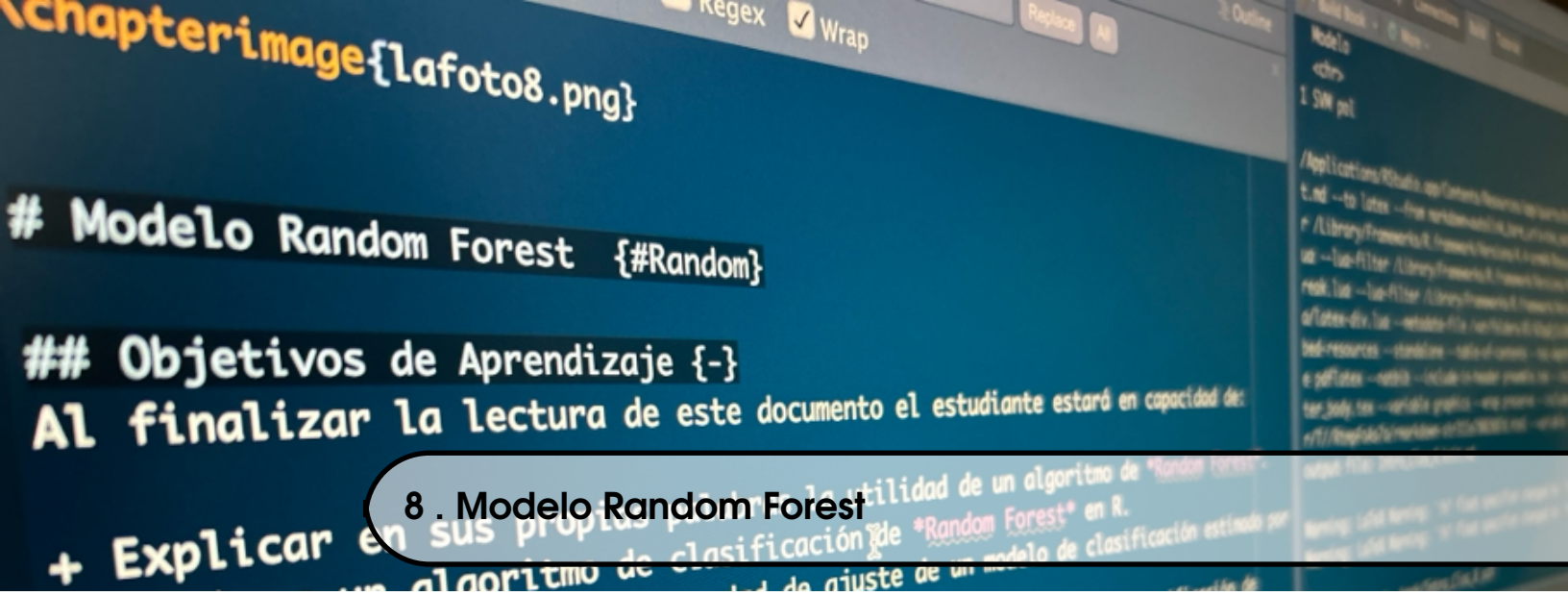
	Accuracy	Sensitivity	Specificity	Precision	F1
Modelo Logit	0.846	0.618	0.873	0.366	0.460
Naive Bayes	0.854	0.472	0.899	0.357	0.407
kNN	0.897	0.167	0.984	0.551	0.256
RT_gini_4niveles	0.906	0.177	0.992	0.728	0.285
RT_gini_500obs	0.907	0.203	0.991	0.727	0.318
RT_gini_podado	0.909	0.254	0.986	0.689	0.371
RT_info_4niveles	0.906	0.177	0.992	0.728	0.285
RT_info_500obs	0.907	0.203	0.991	0.727	0.318
RT_info_podado	0.909	0.254	0.986	0.689	0.371

Fuente: elaboración propia.

7.5 Comentarios Finales

En este Capítulo hemos explorado el método de árbol de decisión aplicado a tareas de clasificación. Como se mencionó anteriormente, *a priori*, es imposible saber si este método será mejor que el estadístico u otros de aprendizaje de máquina y por eso es una buena práctica que se empleen diferentes aproximaciones y se comparen fuera de muestra.

Las aplicaciones de estos métodos de clasificación son muchas en el mundo de los negocios y definitivamente estos modelos son necesarios en la caja de herramientas de un científico de datos.



8 . Modelo Random Forest

Objetivos de Aprendizaje

Al finalizar la lectura de este documento el estudiante estará en capacidad de:

- Explicar en sus propias palabras la utilidad de un algoritmo de *Random Forest*.
- Aplicar un algoritmo de clasificación de *Random Forest* en R.
- Calcular métricas de la bondad de ajuste de un modelo de clasificación estimado por medio de un algoritmo de *Random Forest*.
- Encontrar los pronósticos de clasificación de un algoritmo de clasificación de *Random Forest*.

8.1 Introducción

Hasta ahora, hemos visto 4 métodos para realizar la tarea de clasificación, dos correspondientes al campo estadístico (Modelo *Logit* en el Capítulo 3) y *Naive Bayes* en el Capítulo 5) y dos al campo de la inteligencia artificial (kNN en el Capítulo 6 y Árboles de clasificación en el Capítulo 7). Ahora exploraremos otro método que parte de la idea vista en el capítulo anterior sobre los árboles de clasificación.

Random Forest es una técnica de aprendizaje automático que se basa en la combinación de múltiples árboles de decisión durante el proceso de entrenamiento. Cada árbol de decisión en el bosque se entrena de manera independiente utilizando una muestra aleatoria de los datos de entrenamiento y una selección aleatoria de las características. Luego, para realizar una predicción, el *Random Forest* promedia las predicciones de todos los árboles individuales[1]. Es decir, esta aproximación se basa en la idea que "combinar la sabiduría de muchos árboles" (¡cientos o miles!) es mejor que un solo árbol para obtener una predicción más confiable. La idea detrás de esta aproximación es que la diversidad entre los árboles puede llevar a descubrir patrones más sutiles.

[1] En el caso de la clasificación, se utiliza un voto mayoritario.

A pesar de que un bosque puede contener cientos de árboles, "cultivar" un bosque de árboles de decisión es quizás incluso más fácil que crear un solo árbol altamente afinado. En la siguiente sección discutiremos la intuición detrás del algoritmo.

8.2 Detalles del algoritmo

Para construir un modelo de *Random Forest* se emplea una técnica denominada *bagging*. El *bagging* es una técnica de ensamblaje que combina múltiples modelos de aprendizaje automático entrenados en diferentes conjuntos de datos de entrenamiento para mejorar la precisión y la robustez del modelo final. En este contexto, el *bagging* se utiliza para combinar los resultados de múltiples árboles de decisión entrenados en conjuntos de datos de entrenamiento diferentes. El proceso de *bagging* implica entrenar múltiples árboles de decisión con diferentes muestras de datos de entrenamiento y luego "promediar" las predicciones de todos los árboles para obtener la predicción final del Bosque Aleatorio. Al combinar los resultados de múltiples árboles de decisión entrenados en conjuntos de datos diferentes, el *bagging* ayuda a reducir el *overfitting* y a mejorar su capacidad para generalizar a individuos que no se han observado aún.

Esta técnica implica que los árboles de decisión se entrenan en subconjuntos de muestras aleatorias. Las muestras aleatorias se construyen empleando la técnica de *bootstraping* en la que se crean muestras de datos aleatorias con reemplazo del conjunto de datos original. En la Figura 8.1 se presenta una ilustración de cómo funciona la técnica de *bootstraping*. Partimos de una muestra de tamaño n (recuadro superior gris de la Figura 8.1); cada individuo está representado por un circulo de diferente color. Posteriormente se generan muchas muestras (Nb) aleatorias (cada una de tamaño n) a partir de la muestra original. En cada una de las muestras de *bootstrapping* típicamente se repiten algunos individuos (punto de colores) porque se emplea un re-muestreo con reposición[2].

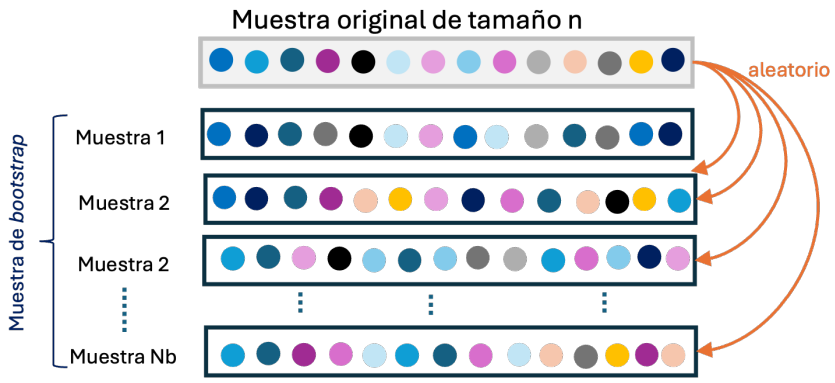
Emplear las muestras de *bootstraping* ayuda a reducir la posible "relación" entre los diferentes árboles que se generen y a mejorar la diversidad del bosque. Luego, todos los árboles "votan" y la mayoría será la clase a la que pertenece la observación.

Además del re-muestreo por *bootstrap*, el *Random Forest* también utiliza una selección aleatoria de variables (en este contexto se emplea más comúnmente el término características) en cada nodo de cada árbol. Esto ayuda a garantizar que cada árbol se especialice en diferentes aspectos de los datos, lo que nuevamente mejora la diversidad del bosque.

Una gran ventaja del *bagging* frente a los árboles individuales es que disminuye la variación del modelo. Los árboles individuales son muy propensos al *overfitting* y son muy sensibles al ruido en los datos. Si los árboles individuales no están correlacionados, combinarlos con el *bagging* los hará más robustos sin aumentar el sesgo.

[2]Note que si no se repitieran los individuos (puntos de color) de la muestra original, entonces tendríamos la muestra original. Esto no tendría sentido, pues el resultado del árbol sería el mismo.

Figura 8.1. Muestras generadas por bootstraping



Fuente: elaboración propia.

8.3 Implementación en R

Para realizar un ejemplo práctico en R (R Core Team, 2023) emplearemos la misma base de datos que empleamos en los capítulos anteriores. Recuerda que la pregunta de negocio que queremos responder en este caso es: ¿se puede construir un modelo que pueda predecir si un cliente adquirirá o no el producto bancario?

Carga el *working space* que guardaste en el Capítulo 7. Para el detalle del preprocesamiento de la muestra ver el Capítulo 3.

Podemos emplear la función **randomForest()** del paquete *randomForest* (Liaw y Wiener, 2002). Esta función en su forma más básica solo requiere de la fórmula y la base de datos.

Hay que recordar que, debido a la naturaleza aleatoria del bosque, los resultados pueden variar ligeramente cada vez que crea el bosque. Por eso es importante emplear una semilla para la generación de números aleatorios con la función **set.seed()** para fijar la semilla. Adicionalmente, recordemos que en el objeto `formula_tree` tenemos la fórmula en la que la variable `y` depende de todas las características disponibles. En este caso el código será:

```
# Cargar Paquete
library(randomForest)
# Fijar la semilla del número aleatorio
set.seed(123456)
# Entrenar el modelo de Random Forest empleando la fórmula que incluye todas
# las variables explicativas en la base de datos

RF_res <- randomForest(formula = y ~ ., data = datos_dummies_est)

RF_res
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = datos_dummies_est)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 10.3%
## Confusion matrix:
##        no  yes class.error
## no  28550  635  0.02175775
## yes  2760 1005  0.73306773
```

Ahora, generemos las predicciones para la muestra de evaluación a partir del modelo de *Random Forest* que acabamos de entrenar. Para esto podemos emplear la función **predict()**.

```
# Generar las predicciones fuera de muestra a partir del modelo Random Forest

pred_RF_res <- predict(RF_res, newdata = datos_dummies_eval, type = "class")
```

Ahora evaluemos el comportamiento fuera de muestra del modelo *Random Forest* empleando la función **confusionMatrix()**, como lo hemos realizado en los capítulos anteriores.

```
mo_RF <- confusionMatrix(pred_RF_res, y_eval, positive = "yes")
mo_RF
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  7217  652
##        yes  146  223
##
##                Accuracy : 0.9031
##                  95% CI : (0.8965, 0.9094)
##     No Information Rate : 0.8938
##     P-Value [Acc > NIR] : 0.002826
##
##                   Kappa : 0.3154
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.25486
##             Specificity : 0.98017
##          Pos Pred Value : 0.60434
##          Neg Pred Value : 0.91714
```

```
##              Prevalence : 0.10622
##          Detection Rate : 0.02707
##    Detection Prevalence : 0.04479
##       Balanced Accuracy : 0.61751
##
##        'Positive' Class : yes
##
```

Ahora podemos comparar esta aproximación con lo obtenido con los otros modelos empleados hasta ahora. En el Cuadro 8.1 se presentan las métricas de todos las aproximaciones estudiadas hasta ahora.

Cuadro 8.1. Métricas de comparación para todos los modelos de clasificación

	Accuracy	Sensitivity	Specificity	Precision	F1
Modelo Logit	0.846	0.618	0.873	0.366	0.460
Naive Bayes	0.854	0.472	0.899	0.357	0.407
kNN	0.897	0.167	0.984	0.551	0.256
RT_gini_500obs	0.907	0.203	0.991	0.727	0.318
RT_gini_podado	0.909	0.254	0.986	0.689	0.371
Random Forest	0.903	0.255	0.980	0.604	0.359

Fuente: elaboración propia.

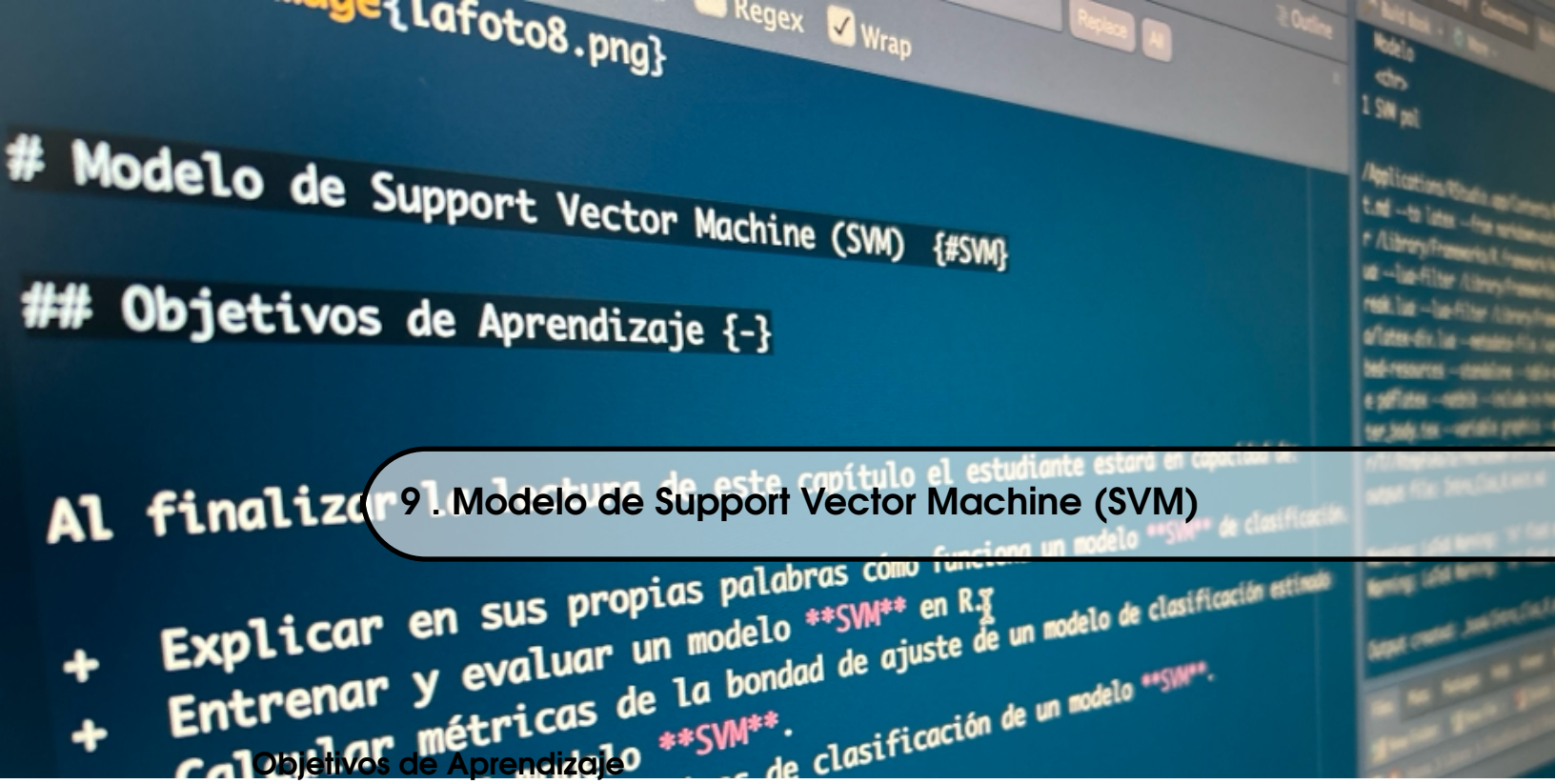
En este caso el *Random Forest* no presenta un mejor comportamiento que los árboles individuales, a excepción de la métrica de *Especificidad*. Una de las limitaciones de esta aproximación es que no podemos graficar el bosque.

Finalmente, guarda el espacio de trabajo para ser empleado en el siguiente capítulo.

8.4 Comentarios Finales

En este capítulo hemos explorado el método de aprendizaje supervisado comúnmente empleado en las tareas de clasificación llamado *Random Forest*, que es el entrenamiento en diferentes muestras de árboles de decisión. Ya sabemos que *a priori* es imposible saber si este método será mejor que el estadístico u otros de aprendizaje de máquina y por eso es común que se empleen diferentes aproximaciones y se comparen. No te dejes confundir por el resultado específico obtenido en el ejercicio de este capítulo. Este resultado es específico a esta muestra y la semilla aleatoria empleada.

Las aplicaciones de estos métodos de clasificación son muchas en el mundo de los negocios y definitivamente estos modelos son necesarios en la caja de herramientas de un científico de datos.



9. Modelo de Support Vector Machine (SVM)

Objetivos de Aprendizaje

Al finalizar la lectura de este capítulo el estudiante estará en capacidad de:

- Explicar en sus propias palabras cómo funciona un modelo **SVM** de clasificación.
- Entrenar y evaluar un modelo **SVM** en R.
- Calcular métricas de la bondad de ajuste de un modelo de clasificación estimado por medio de un modelo **SVM**.
- Encontrar los pronósticos de clasificación de un modelo **SVM**.

9.1 Introducción

En los capítulos anteriores hemos discutido modelos de clasificación de aprendizaje de máquina con filosofías relativamente diferentes. El modelo kNN (Capítulo 6) emplea los k vecinos más cercanos para votar sobre la clase de un individuos. El algoritmo de clasificación de árbol (Capítulo 7) construye el modelo de clasificación a través de la creación de divisiones jerárquicas basadas en las variables consideradas. Estos enfoques, diferentes en su filosofía, ayudan a abordar la tarea de clasificación desde diferentes perspectivas. Por otro lado, el *Random Forest* es un algoritmo que continúa con la idea de los árboles de clasificación y combina múltiples árboles de decisión para mejorar la precisión y generalización del modelo. Cada árbol en el bosque se entrena de forma independiente y luego se combinan los resultados para obtener una predicción más robusta. Estas aproximaciones, combinadas con las aproximaciones de origen estadístico como el modelo *Logit* (Capítulo 3) y el **Naive Bayes** (Capítulo 5), se complementan para atacar la tarea de clasificación desde diferentes frentes.

En este Capítulo estudiaremos un método del campo de *machine learning* que tiene una filosofía totalmente diferente: el modelo *Support Vector Machine* (**SVM**)[1]. El modelo **SVM** entra a completar nuestra caja de herramientas de métodos para realizar la

[1] De hecho, el modelo **SVM** permite realizar tanto la tarea de clasificación como la de regresión.

tarea de clasificación. Estos modelos se caracterizan por comportarse relativamente bien cuando se cuenta con muchas variables explicativas (*features*), es decir, datos de alta dimensión y con comportamientos no lineales. Estas dos características hacen del modelo **SVM**, un modelo versátil para resolver una amplia gama de problemas de clasificación (Wang et al., 2009).

Los modelos **SVM** se han convertido en un método potente y ampliamente utilizado para resolver problemas de clasificación en diferentes campos, como el reconocimiento de imágenes, la categorización de textos, la bioinformática y el marketing. En la siguiente sección estudiaremos intuitivamente cómo funciona este modelo para después realizar una aplicación en R.

9.2 El método

El modelo **SVM** tiene como principio intentar separar lo mejor posible las diferentes clases empleando un "punto", línea o hiperplano[2]. Empleemos unos ejemplos para entender esta idea.

Partamos de un caso muy sencillo en el que tenemos una variable objetivo con dos categorías, tal como lo hemos hecho en los capítulos pasados. Tenemos unas observaciones que corresponden a negativos (puntos rojos) y otras a positivos (puntos verdes) y una sola variable explicativa (*feature*) X_1, tal como se muestra en el panel a) de la Figura 9.1.

En este contexto, el objetivo del modelo de clasificación, es poder "clasificar" una nueva observación en negativo y positivo empleando una única variable con la que contamos para entrenar el modelo (X_1). Es decir, tenemos un problema de una dimensión. Para este caso de un *feature*, la filosofía de este modelo, implica encontrar un "punto" para los valores de X_1 a partir del cual se puede determinar que el individuo se clasificará como positivo. En el panel b) de la Figura 9.1 se presenta un punto (en color naranja) que clasifica bien a todos los individuos.

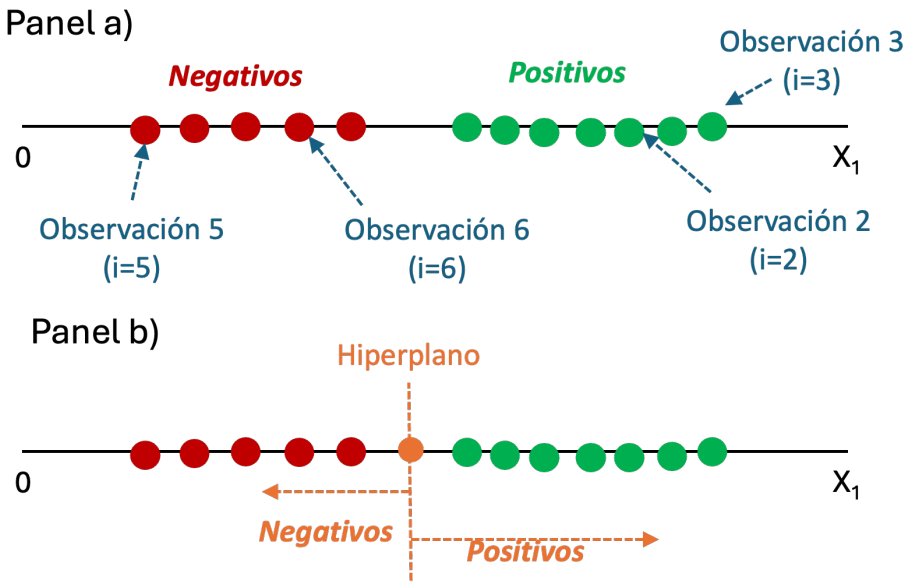
Consideremos el caso en el que se cuenta con dos características (X_1 y X_2) para clasificar a los individuos en positivos (puntos verdes) y negativos (puntos rojos) como se muestra en la Figura 9.2. En el panel b) de la Figura 9.2 se puede ver cómo el método **SVM** implicará encontrar una línea (línea naranja) para clasificar las observaciones positivas y negativas. En este caso, todos los individuos han quedado bien clasificados.

Ahora, supongamos que tenemos 3 variables explicativas (X_1, X_2 y X_3). En este caso, el algoritmo **SVM** tendrá que emplear un plano para clasificar los datos (Ver Figura 9.2). Y así sucesivamente, en dimensiones mayores (más variables) se empleará el equivalente a la línea o el plano para clasificar los individuos.

En general, los modelos de **SVM** pueden entenderse a partir del concepto de hiperplano. Un hiperplano es un límite de decisión que separa puntos de datos de diferentes clases en un espacio de características. El objetivo del modelo **SVM** es encontrar el

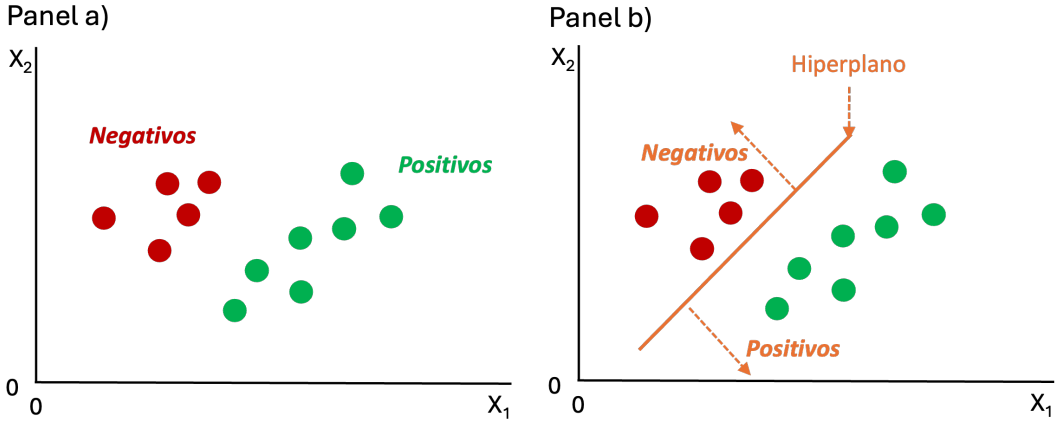
[2]Dependiendo de la cantidad de variables explicativas (*features*) que se empleen corresponderá a una línea para dos variables, un plano para tres variables, etc. Es decir, un hiperplano.

Figura 9.1. Representación gráfica del modelo SVM para un espacio de una característica (una variable explicativa)



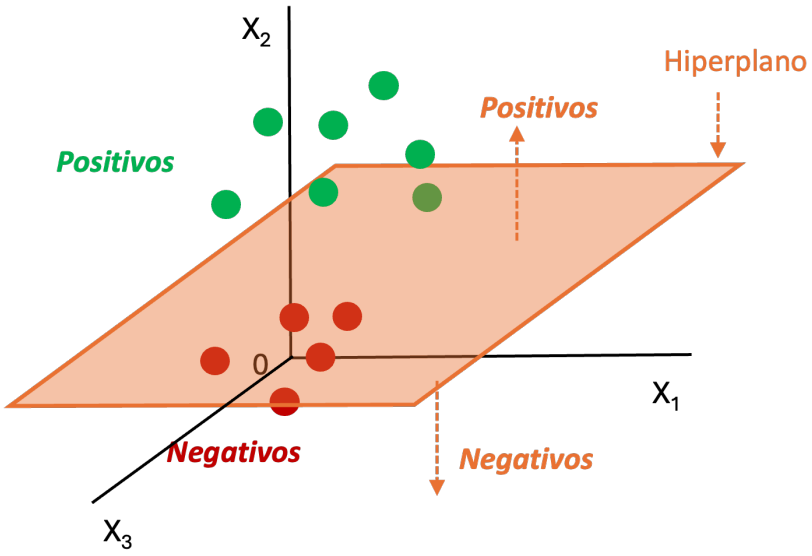
Fuente: elaboración propia.

Figura 9.2. Representación gráfica del modelo SVM para un espacio de dos características (dos variables explicativas)



Fuente: elaboración propia.

Figura 9.3. Representación gráfica del modelo SVM para un espacio de tres características (tres variables explicativas)



Fuente: elaboración propia.

hiperplano que separa al máximo los puntos de datos de las diferentes clases. Regresemos al caso de dos características, note que se podrían trazar muchas líneas rectas que separen los datos (Ver panel a) de la Figura 9.3). Para encontrar el hiperplano de separación óptimo, el **SVM** emplea un proceso de optimización.

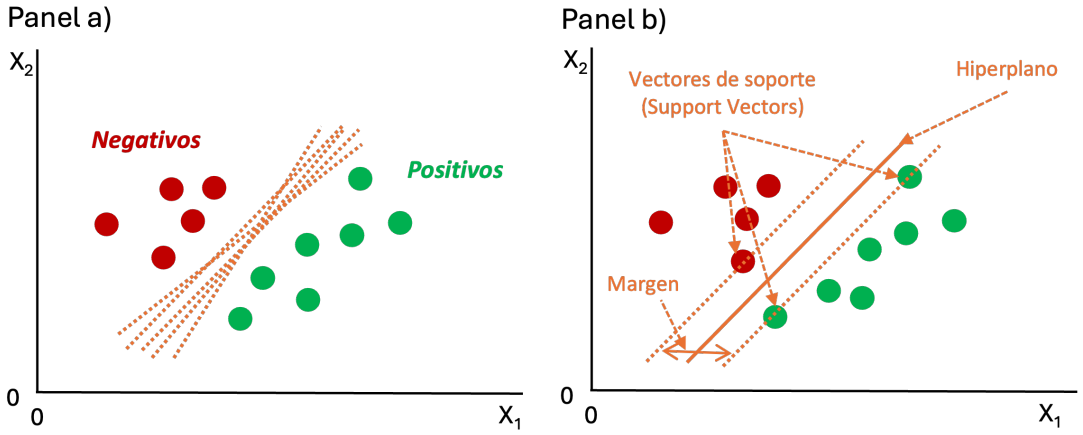
El objetivo de un **SVM** es encontrar el hiperplano que maximice el margen entre ambas clases. El margen es la distancia mínima entre el hiperplano y los puntos de datos más cercanos a él (Ver panel b) de la Figura 9.4). Los puntos más cercanos son de hecho el soporte del hiperplano, de ahí que esos puntos son llamados vectores de soporte en inglés *support vector*, lo cuál da el nombre al método (Ver panel b) de la Figura 9.4).

El margen, entonces, corresponde a la distancia entre el hiperplano y los puntos de datos más cercanos de cada clase. Un margen más amplio indica un modelo más robusto y generalizable, ya que hay un mayor espacio entre las clases. Al optimizar el margen de un modelo *SVM*[3] se puede lograr un equilibrio entre precisión para la muestra de entrenamiento y la generalización a cualquier nueva muestra, incluyendo la de evaluación.

En esencia, el hiperplano actúa como una frontera de decisión, clasificando nuevos datos a un lado u otro del mismo y el margen da una zona a lado y lado del hiperplano en la que se pueden encontrar los puntos de datos de las diferentes clases. Este espacio adicional proporcionado por el margen ayuda a mejorar la capacidad del modelo

[3]Maximizar el margen implica maximizar la distancia entre los vectores de soportes y el hiperplano.

Figura 9.4. Diferentes posibles hiperplanos y el margen en el modelo SVM



Fuente: elaboración propia.

SVM para generalizar de manera efectiva a nuevos individuos aún no observados y realizar predicciones mas precisas.

¡Pero el mundo no necesariamente es lineal! En algunas ocasiones podríamos tener datos como los que se presentan en el panel a) de la Figura 9.5. En un caso como este, el **SVM** no hará una buena clasificación de los individuos como si ocurrió en los casos mostrados en las Figuras 9.1), 9.2) y 9.3). Si empleamos un hiperplano (que es lineal), en estos datos tendremos puntos mal clasificados y algunas violaciones del margen (Ver panel b) de la Figura 9.5).

En algunas ocasiones los datos no presentan una clasificación clara, o por lo menos que se pueda realizar con un plano (lineal) (como el caso del panel b) de la Figura 9.5). En las Figuras 9.1 y 9.2 se muestran casos en los que los datos se pueden separar fácilmente con un hiperplano. En ese caso, se dice que los datos son linealmente separables. En las Figuras 9.5 y 9.6 se presentan ejemplos de datos que no son linealmente separables.

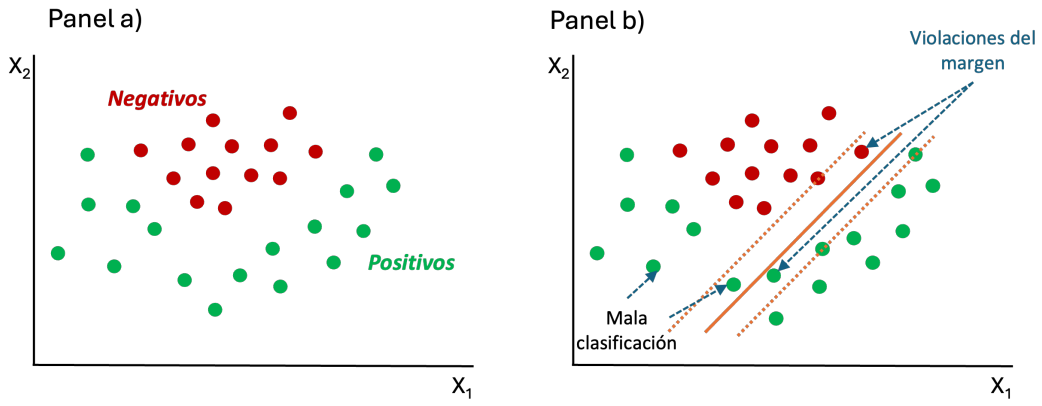
Para resolver este problema, es importante introducir un nuevo concepto: el *kernel*. El *kernel* es una función matemática que permite mapear (transformar) los datos originales a un espacio de características (variables) diferentes, donde la separación entre clases se vuelve más evidente[4].

El hiperplano[5] separa las clases, y el *kernel* es una herramienta que permite transformar los datos a un espacio diferente donde esa separación es más clara que permite

[4]En el contexto estadístico a la transformación de variables se le conoce como reparametrizar. En el Capítulo 2 de Alonso (2024) se presenta una introducción a este concepto.

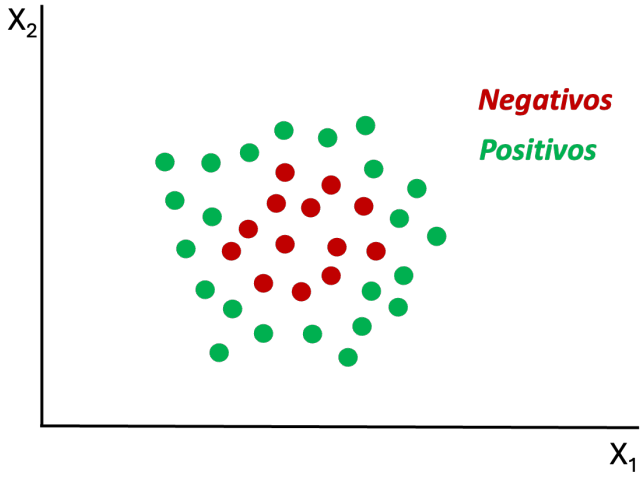
[5]Como lo dice su nombre, el hiperplano es plano; es decir, lineal.

Figura 9.5. Datos no lineales y el modelo SVM



Fuente: elaboración propia.

Figura 9.6. Ejemplo de datos no separables lienalmente

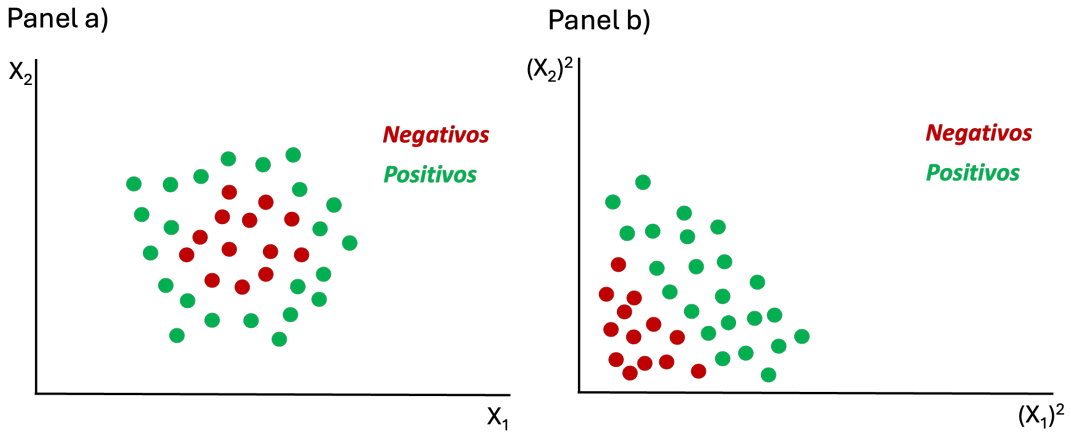


Fuente: elaboración propia.

a los **SVM** trabajar con datos no linealmente separables, mientras que el hiperplano es la frontera que realiza la clasificación[6].

Por ejemplo, podemos emplear una transformación como elevar al cuadrado las dos variables para convertir los datos del panel a) de la Figura 9.7 para obtener unos datos linealmente separables como se presenta en el panel b) de la misma figura. Es decir, en el panel b) de la Figura 9.7 se presentan los datos del panel después de aplicarles el *kernel*. Al trabajar en este nuevo espacio[7], ahora se puede encontrar un hiperplano lineal que separe mejor las clases, lo cual no era posible en el espacio original.

Figura 9.7. Ejemplo de datos no separables linealmente transformados con un kernel a datos separables



Fuente: elaboración propia.

Existen diferentes tipos de *kernels*, cada uno con sus propias ventajas y desventajas. Los *kernels* más comunes son: lineal, polinómico y el **RBF** (*Radial Basis Function*)[8]. Formalmente,

Kernel Lineal: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$ **Kernel Polinómico**: $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma(\mathbf{x}_i^T \mathbf{x}_j + c_0))^d$ **Kernel RBF**: $K(\mathbf{x}_i, \mathbf{x}_j) = e^{\gamma ||\mathbf{x}_i^T \mathbf{x}_j||^2}$ Noten que d corresponde al grado de la transformación polinómica y c_0 controla cuánto influyen los términos de grado más alto en el *kernel* polinómico. Un valor de c_0 más alto dará más peso a los términos de grado superior, lo que puede llevar a un límite de decisión menos lineal y flexible. Por otro lado, un valor más bajo de c_0 reduce la influencia de los términos de grado superior, lo que hace que

[6]Si estas familiarizado con el modelo de regresión clásico, de pronto te será común hacer transformaciones de variables para convertir un modelo que no es lineal matemáticamente a uno lineal desde el punto de vista estadístico (Ver Capítulo 2 de Alonso (2024)). Por ejemplo, cuando se emplean logaritmos para linealizar una función Cobb-Douglas. Es decir, linealizar un modelo en regresión múltiple es equivalente a lo que hace la función *kernel* para el modelo **SVM**.

[7]Es decir, trabajar con las variables explicativas transformadas.

[8]La forma funcional del *kernel* simula el principio del algoritmo kNN. Es decir, genera una transformación que cuanto más cerca estén dos puntos entre sí en términos de atributos, más probable es que sean similares.

el límite de decisión sea más suave y lineal. Y γ controla la influencia de un solo punto de datos en el ajuste del modelo. Valores bajos de γ significan que los puntos de datos lejanos tienen un impacto grande en el límite de decisión, mientras que valores altos significan que solo los puntos cercanos tienen un impacto significativo.

El *kernel* **RBF** es mucho más flexible que los otros dos *kernels* pues permite ajustarse a transformaciones muy complejas. Por esta razón, esta es la transformación más empleada en la práctica.

Por otro lado, al entrenar un modelo **SVM** también se debe escoger el parámetro de costo. Este parámetro representa la importancia de clasificar correctamente cada punto de datos en el proceso de entrenamiento. En otras palabras, este parámetro controla el equilibrio entre maximizar el margen y minimizar la clasificación incorrecta de puntos. Un valor alto del costo significa que el modelo intentará clasificar correctamente la mayoría de los puntos de datos de entrenamiento, incluso si eso significa tener un margen más estrecho. Por otro lado, un valor bajo del costo permitirá un margen más amplio, lo que puede resultar en una clasificación incorrecta de algunos puntos de datos de entrenamiento, pero posiblemente una mejor generalización para datos nuevos. En otras palabras, el parámetro de costo establece cuánto error estamos dispuestos a sacrificar para encontrar el hiperplano. Esto se hace con el fin de evitar el *overfitting* o sobreajuste, además de disminuir la sensibilidad del método ante nuevas observaciones.

Así, en la práctica nos enfrentaremos a decidir:

- el *kernel* a usar,
- el costo,
- d y c_0 para el *kernel* polinómico,
- γ para los *kernels* polinómico y **RBF**.

Por lo general, se entrena el modelo **SVM** para los diferentes *kernels* empleando una búsqueda de grilla para los parámetros pertinentes. Esa búsqueda de los mejores parámetros se denomina en inglés *tune* y en la jerga de los científicos de datos como "tunear" el modelo. Y finalmente se selecciona la mejor combinación de parámetros para un **SVM** y *kernel* empleando una métrica de selección, por ejemplo el *Accuracy*.

9.3 Implementación en R

Empecemos por cargar el *working space* que guardaste en el Capítulo 8.

En este caso, similar a lo ocurrido en el modelo kNN (Ver Capítulo 6), es necesario usar las variables cuantitativas estandarizadas[9] (las de clase numérico (**numeric**) o entero (**integer**) para evitar que las escalas diferentes tengan un efecto sobre el algoritmo.

Como se discutió en la Sección 6.3, en este caso las variables a estandarizar son:

[9] Es decir, se le quita la respectiva media para centrar los datos y se divide por la desviación estándar para garantizar que la varianza sea uno.

- `age`,
- `campaign`,
- `previous`,
- `emp.var.rate`,
- `cons.price.idx`,
- `cons.conf.idx`,
- `euribor3m`.

En la Sección 6.3 ya realizamos dicha estandarización empleando la función **scale()** de la base de R. Emplearemos los objetos `datos_expl_est` y `y_est` en los que tenemos las variables explicativas para la muestra de estimación y la variable objetivo para la muestra de evaluación respectivamente (Ver sección 6.3).

Por otro lado, en el modelo **SVM** se pueden emplear variables predictoras (*features*) cualitativas codificadas como variables dummy (de clase **factor**) tal como lo realizamos en el Capítulo 3. En ese Capítulo empleamos la función **dummy_cols()** del paquete *fastDummies* (Kaplan, 2023) para crear las correspondientes variables dummy que hemos venido empleando en los siguientes capítulos.

Dichas variables dummy las tenemos en la columna 10 a la 50 de los objetos `datos_dummies_est` y `datos_dummies_eval` que corresponden a las muestras de estimación y evaluación respectivamente.

Procedamos a unir las variables predictoras estandarizadas y las variables en un solo objecto, uno para la muestra de entrenamiento y otra para la de evaluación.

```
# Crear data.frame con variables cuantitativas escaladas y variables dummies
datos_X_svm_est <- bind_cols(datos_expl_est, datos_dummies_est[, 10:50])
datos_X_svm_eval <- bind_cols(datos_expl_eval, datos_dummies_eval[, 10:50])

# Unir las bases en una sola para la muestra de estimación

datos_svm_est <- cbind(datos_X_svm_est, y_est)
```

Para realizar el entrenamiento del modelo **SVM** en R, usaremos el paquete *caret*(Kuhn y Max, 2008) que permite "tunear" el modelo empleando tres funciones: **trainControl()**, **expand.grid()** y **train**. La función **trainControl()** permite establecer las condiciones bajo las cuales se realizará el ejercicio de **tunning**. La función **expand.grid()** establece sobre que valores se va a desarrollar la búsqueda de grilla; y finalmente la función **train()** pone todo junto y establece el tipo de modelo que se empleará, así como los datos.

Realicemos un ejercicio de "tuneo" del modelo que realice una búsqueda de grilla (**search = "grid"**)[10] y que emplee *bootstrapping*[11] para evaluar sobre diferentes submuestras el desempeño de todas las combinaciones de parámetros a evaluar.

[10] Otra opción de este argumento es que haga la búsqueda de manera aleatoria (**search = "random"**).
[11] Para una explicación del método de *bootstraping* ver la discusión de la sección 8.2.

```
# Cargar la librería

library(caret)


# Fijar la forma como se realizará el tunninng
metodo_tune <- trainControl(method = "boot", search = "grid")
```

Dado que queremos entrenar un modelo **SVM** con *kernel* lineal, solo necesitamos realizar una búsqueda de grilla sobre el parámetro costo. Consideremos una búsqueda sobre los siguientes valores de costo (C): 0.01, 0.1, 1, 10 y 50. Esto lo podemos hacer con el siguiente código.

```
# Establecer la grilla de búsqueda para el costo
grilla_busqueda <- expand.grid(C = c(0.01, 0.1, 1, 10, 50))
```

Ya tenemos los objetos `metodo_tune` y `grilla_busqueda` en los que hemos especificado el tipo de ejercicio de *tunning* y la grilla sobre la que se hará el ejercicio, respectivamente. Ahora, solo necesitamos entrenar el modelo **SVM** empleando la función **train()**.

Similar a las funciones empleadas en capítulos anteriores, el primer argumento de la función es la fórmula (**form**) que específica las variables dependientes e independiente. Adicionalmente, está el argumento para los datos (**data**) que deben estar en un objeto de clase **data.frame** . Otro argumento es **method** que específica el tipo de modelo a entrenar. Para el caso de un **SVM** con *kernel* lineal, este argumento se convierte en **method = "svmLinear"**. Las funciones que estiman el modelo **SVM** en *caret* pertenecen al paquete *kernlab* . Y por tanto, necesitarás tener instalado dicho paquete para poder entrenar el modelo **SVM**. Otros dos argumentos son **trControl** y **tuneGrid** que especifican el tipo de ejercicio de *tunning* que se realizará y el espacio sobre el que se realizará la búsqueda de grilla, respectivamente. Finalmente, es necesario especificar cuál métrica (*metric*) se empleará para seleccionar la mejor combinación de parámetros.

Para nuestro caso, entonces emplearemos como fórmula a la variable `y` en función de todas las variables en el **data.frame** `datos_svm_est` (nuestra muestra de estimación). Emplearemos la especificación del ejercicio de *tunning* que guardamos en el objeto `metodo_tune`, el **accuracy** como métrica de selección del mejor modelo y una búsqueda de grilla sobre los posibles valores de costo definidos en el objeto `grilla_busqueda`. Todo esto lo logramos con el siguiente código:

```
set.seed(123)

## empezar el tunning para el modelo svm con kernel lineal
model_svm_lineal <- train(form = y_est ~ ., data = datos_svm_est, method =
↪  "svmLinear",
    trControl = metodo_tune, metric = "Accuracy", tuneGrid = grilla_busqueda)
```

Estos cálculos tomarán un tiempo considerable. Dependiendo de tu computador, el cálculo podrá tomar hasta días, ¡ten paciencia! En el objeto `model_svm_lineal` quedará guardada toda la búsqueda y el mejor modelo (combinación de parámetros) de acuerdo a la métrica seleccionada, así mismo quedará guardado en este objeto muchos otros resultados del proceso de búsqueda. Veamos el objeto `model_svm_lineal` y todos sus compartimientos:

```
# ver el objeto
model_svm_lineal
```

```
## Support Vector Machines with Linear Kernel
##
## 32950 samples
##    48 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 32950, 32950, 32950, 32950, 32950, 32950, ...
## Resampling results across tuning parameters:
##
##   C      Accuracy   Kappa
##    0.01  0.8869937  0.1302668
##    0.10  0.8880890  0.1925624
##    1.00  0.8880822  0.1934050
##   10.00  0.8864819  0.1837770
##   50.00  0.8863731  0.2046663
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.1.
```

```
# inspeccionar todos los compartimientos del objeto
attributes(model_svm_lineal)
```

```
## $names
##  [1] "method"       "modelInfo"    "modelType"    "results"      "pred"
##  [6] "bestTune"     "call"         "dots"         "metric"       "control"
## [11] "finalModel"   "preProcess"   "trainingData" "ptype"        "resample"
## [16] "resampledCM"  "perfNames"    "maximize"     "yLimits"      "times"
## [21] "levels"       "terms"        "coefnames"    "xlevels"
##
## $class
## [1] "train"         "train.formula"
```

Los parámetros que optimizan la métrica seleccionada quedan guardados en el compartimiento con el nombre `bestTune`. Para nuestro caso, veamos el mejor valor del costo con el siguiente código.

```
# mirar el valor de los parámetros óptimo
model_svm_lineal$bestTune
```

```
##     C
## 2 0.1
```

Ahora podemos realizar la misma tarea empleando el *kernel* polinómico. Ahora, deberos modificar el espacio sobre el que se realizará la búsqueda de grilla para incorporar, además del costo, el grado del polinomio (**degree**) y la escala (**scale**).

```
# Establecer la grilla de búsqueda para el modelo SVM kernel polimonial
grilla_busqueda <- expand.grid(C = c(0.01, 0.1, 1, 10), degree = c(2, 3), scale
↪  = 0.01)
```

Ahora podemos emplear la función **train()** para entrenar el modelo **SVM** con *kernel* polinómico empleando **method = svmPoly**.

```
# Tunear el modelo SVM kernel polimonial

model_svm_pol <- train(y_est ~ ., data = datos_svm_est, method = "svmPoly",
↪  trControl = metodo_tune,
    metric = "Accuracy", tuneGrid = grilla_busqueda)
```

Este proceso de *tunning* tomará un poco menos de tiempo que el caso anterior. Los parámetros que optimizan el *accuracy* son:

```
# parámetros que optimizan el accuracy del modelo SVM kernel polinómico

model_svm_pol$bestTune
```

```
##   degree scale  C
## 7      2  0.01 10
```

Y finalmente realicemos el mismo ejercicio empleando el *kernel* **RBF**. En este caso, debemos modificar la grilla de búsqueda de la siguiente manera:

```
# Establecer la grilla de búsqueda para el modelo SVM kernel RBF

grilla_busqueda <- expand.grid(C = c(0.01, 0.1, 1, 10), sigma = c(0.01, 0.1,
↪  1))
```

Y el modelo **SVM** con *kernel* **RBF** (**method = "svmRadial"**) se puede entrenar con el siguiente código.

```
# Tunear el modelo SVM kernel RBF
model_svm_rbf <- train(y_est ~ ., data = datos_svm_est, method = "svmRadial",
↪  trControl = metodo_tune,
    metric = "Accuracy", tuneGrid = grilla_busqueda)
```

En este caso los parámetros que optimizan el *accuracy* son:

```
# parámetros que optimizan el accuracy del modelo SVM kernel RBF

model_svm_rbf$bestTune
```

```
##   sigma C
## 7  0.01 1
```

Nota que el parámetro de costo que optimiza el *accuracy* en los tres *kernels* considerados es diferente. Es decir, siempre deberíamos hacer la búsqueda de grilla sobre el espacio completo y no podemos emplear el resultado de un *kernel* para los otros. Estos procesos de *tunning* del modelo **SVM** son bastante exigentes en recursos de cómputo. Este último proceso toma aún más tiempo.

9.3.1 Bondad de ajuste del modelo

Ahora evaluemos fuera de muestra el comportamiento del modelo **SVM** para los tres *kernels* considerados. Para esto, podemos emplear las funciones **predict()** y **confusionMatrix()** como lo hemos hecho en los capítulos anteriores. Es decir, en nuestro caso tendremos para el *kernel* lineal el siguiente código:

```
# Calcular las predicciones en la muestra de evaluación
pred_svm_lineal <- predict(model_svm_lineal, datos_X_svm_eval)
# Calcular las métricas
mo_svm_lineal <- confusionMatrix(pred_svm_lineal, y_eval, positive = "yes")
mo_svm_lineal
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  7232  690
##        yes  131  185
##
##                Accuracy : 0.9003
##                  95% CI : (0.8937, 0.9067)
##     No Information Rate : 0.8938
##     P-Value [Acc > NIR] : 0.02705
##
##                   Kappa : 0.2695
##
##  Mcnemar's Test P-Value : < 2e-16
##
##             Sensitivity : 0.21143
##             Specificity : 0.98221
##          Pos Pred Value : 0.58544
##          Neg Pred Value : 0.91290
```

```
##              Prevalence : 0.10622
##          Detection Rate : 0.02246
##    Detection Prevalence : 0.03836
##       Balanced Accuracy : 0.59682
##
##        'Positive' Class : yes
##
```

En el Cuadro 9.1 se presentan las métricas de los modelos **SVM** con los tres *kernels* considerados para la muestra de evaluación.

Cuadro 9.1. Métricas de comparación para todos los modelos de clasificación

	Accuracy	Sensitivity	Specificity	Precision	F1
Modelo Logit	0.847	0.617	0.874	0.368	0.461
Naive Bayes	0.854	0.472	0.899	0.357	0.407
kNN	0.895	0.240	0.973	0.516	0.328
RT_gini_500obs	0.907	0.203	0.991	0.727	0.318
RT_gini_podado	0.909	0.254	0.986	0.689	0.371
Random Forest	0.903	0.255	0.980	0.604	0.359
SVM lineal	0.900	0.211	0.982	0.585	0.311
SVM pol	0.899	0.226	0.979	0.561	0.322
SVM RBF	0.897	0.153	0.986	0.558	0.240

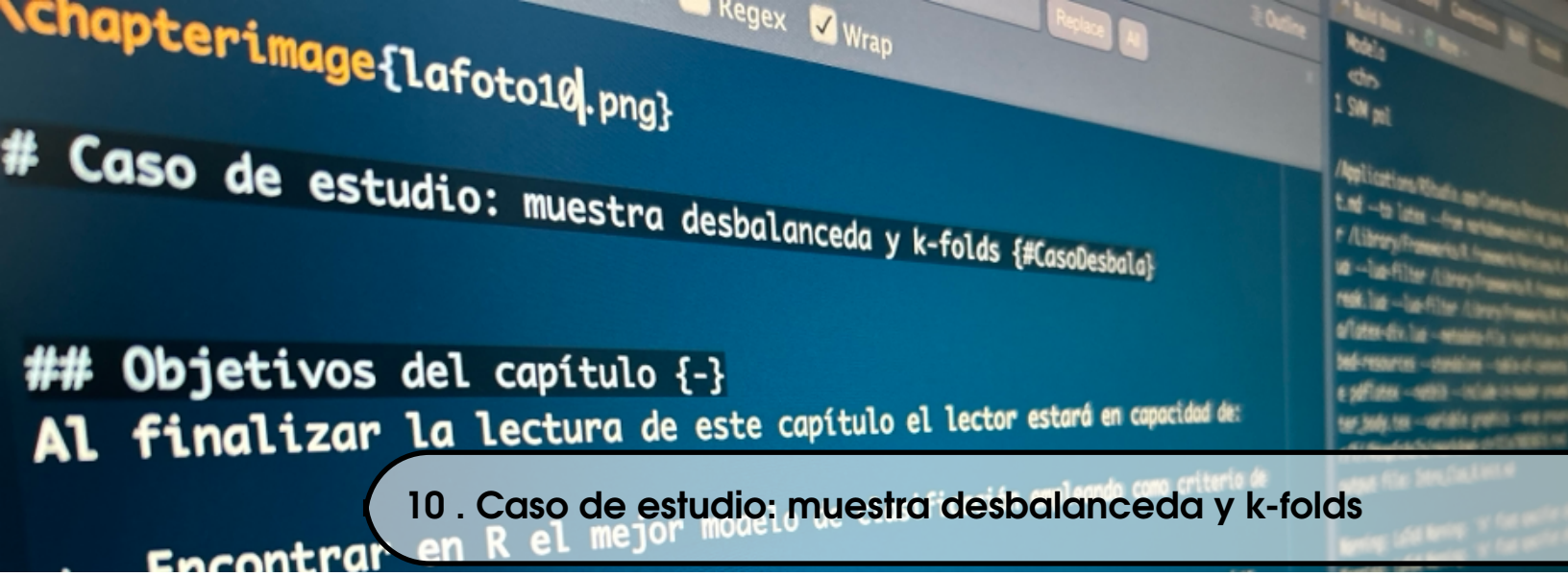
Fuente: elaboración propia.

Poniendo todo junto, tenemos 9 modelos de clasificación candidatos a ser el "mejor" modelo. En términos de *accuracy* es el que hemos denominado RT_gini_podado. En términos de sensibilidad el mejor modelo es Modelo Logit. En especificidad el mejor modelo es el RT_gini_500obs. En precisión el mejor modelo es RT_gini_500obs. Y en términos del puntaje F_1, el mejor modelo es Modelo Logit. Decidir entre estos modelos no es una tarea fácil y tendrá que ser el negocio, en especial el *analytics translator* con los tomadores de decisiones que tendrán que determinar que métrica es mas conveniente para el negocio y el uso que se le dará al modelo seleccionado. Este resultado nos permite entender la importancia de realizar un análisis detallado de cada métrica y su relevancia en el contexto específico de aplicación de los modelos de clasificación.

9.4 Comentarios finales

En este Capítulo hemos estudiado el modelo **SVM** de origen en el *machine learning* para desarrollar la tarea de clasificación. Este modelo completa la caja de herramientas para realizar esta tarea. Aunque existen aún más modelos, estos son los algoritmos más usados por científicos de datos en la actualidad. Finalmente, como se ha reiterado durante este libro, la práctica ideal es implementar todos los modelos para compararlos. Después de tener el mejor modelo, emplear toda la muestra para estar listo a realizar una predicción para individuos nuevos.

En los siguientes dos capítulos desarrollaremos dos casos de estudio en el que emplearemos todos los modelos estudiados hasta ahora.



10 . Caso de estudio: muestra desbalanceda y k-folds

Objetivos del capítulo

Al finalizar la lectura de este capítulo el lector estará en capacidad de:

- Encontrar en R el mejor modelo de clasificación empleando como criterio de validación *k-fold*.
- Encontrar en R el mejor modelo de clasificación a partir de una muestra desbalanceda.

10.1 Introducción

En este capítulo aplicaremos todo lo estudiado hasta el momento. A lo largo del libro hemos desarrollado un ejemplo para datos de varias campañas de *marketing* directo de una institución bancaria portuguesa (Moro et al., 2014). En cada capítulo aplicábamos un modelo diferente de clasificación intentando responder la pregunta de negocios: ¿se puede construir un modelo que pueda predecir si un cliente adquirirá o no el producto bancario?.

Adicionalmente, en cada uno de los capítulos empleamos para realizar la validación cruzada de cada modelo el **método de retención** o *holdout method* (Ver Sección 2.2.1 para una discusión del método). El método de retención implica reservar una proporción de la muestra (en nuestro caso el 20 %) para realizar la evaluación del modelo que es estimado o entrenado con el restante 80 % de la muestra.

Recordemos que es una buena práctica realizar una validación cruzada de un modelo para evitar un posible *overfitting* o sobreajustes del modelo. La validación cruzada nos permite evaluar la capacidad predictiva del modelo de una manera más robusta al probarlo en diferentes conjuntos de datos. En la Sección 2.2 discutimos otras metodologías para hacer validación cruzada, una de ellas (y tal vez la más usada en la práctica) es k **iteraciones** o *k-fold Cross-validation*.

El método *k-folds* implica dividir de manera aleatoria la muestra completa en $kgrupos$ de aproximadamente el mismo tamaño. Para cada uno de los k grupos (o iteraciones) se emplean los restantes $k-1$ grupos como muestra de estimación y el grupo k de observaciones se emplea como muestra de evaluación para la cual se calculan las respectivas métricas deseadas para los modelos a comparar. Y finalmente, para obtener la métrica para todo el ejercicio se calcula el promedio de las k métricas calculadas para cada modelo (Ver Figura 2.3). En este capítulo estudiaremos cómo implementar el método de *k-folds* en R.

Otro tema que hemos evadido en el libro, es el tema de las muestras desbalanceadas. En este Capítulo discutiremos brevemente que opciones tenemos para solucionar el sesgo que puede producir una muestra desbalanceada y cómo implementar esto en R.

10.1.1 Sobre las muestras desbalanceadas

Una muestra desbalanceada, en el contexto de modelos de clasificación en ciencia de datos, se refiere a una muestra en el que las clases que se quieren predecir están representadas por un número desproporcionado de observaciones. Esto significa que una clase puede estar subrepresentada en comparación con otras clases en el conjunto de datos. Recordemos, que como lo discutimos en el Capítulo 3, si la muestra tiene "muy pocos" valores de una de las clases de la variable dependiente se puede producir un problema de sesgo por datos desbalanceados. Dicho sesgo se puede solucionar utilizando técnicas de muestreo para equilibrar las clases. Algunas de estas técnicas incluyen el **submuestreo** de la clase mayoritaria o el **sobremuestreo** de la clase minoritaria. Otra opción es utilizar algoritmos de clasificación que sean menos sensibles al desbalance de clases, como el *Random Forest* (Ver Capítulo 8) o el

Support Vector Machine (Ver Capítulo 9). Cómo solucionar ese problema se discutirá en parte en el Capítulo 10.

En el ejercicio realizado en todos los capítulos hemos empleado una muestra en la que la proporción de clientes que adquirieron el producto era de 11.27%. Si bien es discutible, en el Capítulo 3 argumentamos que esa muestra era relativamente balanceada entre los `no` y los `yes`. Esta afirmación es muy discutible. Algunos autores como Chawla et al. (2002), Weiss y Provost (2003) y García et al. (2010) consideran que una muestra está desbalanceada cuando una clase representa menos del 20% de las observaciones totales. Otros autores como Batista et al. (2004), He y Garcia (2009) y Weiss y Provost (2003) argumentan que el umbral debería ser 10%. No existe un consenso sobre que umbral debería emplearse para determinar cuándo una muestra se considera desbalanceada, pues puede depender del problema específico y del dominio de aplicación.

Para solucionar el problema de una muestra desbalanceada, existen varias técnicas que se pueden aplicar, las cuales se pueden dividir en dos categorías:

1. Modificación del conjunto de datos (muestreo de datos)
2. Modificación del algoritmo (asignación de pesos)

La primera aproximación implica modificar, por muestreo aleatorio, la composición de los datos originales de tal manera que no estén desbalanceados. Entre las técnicas de remuestreo encontramos:

- Submuestreo (*Down-sampling*): Se eliminan aleatoriamente individuos de la clase mayoritaria para equilibrar la distribución de las clases. Sobremuestreo (*Up-sampling*): Se "crean nuevos" individuos de la clase minoritaria para aumentar su tamaño. Existen diferentes técnicas para realizar el sobremuestreo, como la generación de datos "sintéticos" o el remuestreo con reemplazo.
- SMOTE (*Synthetic Minority Over-sampling Technique* o Técnica de muestreo sintético de minorías): Es una técnica de sobremuestreo que genera nuevos ejemplos de la clase minoritaria a partir de sus vecinos más cercanos a la vez que genera un muestreo descendente de la clase mayoritaria.

Por otro lado, entre las técnicas que modifican el algoritmo encontramos el **ajuste de pesos o ponderaciones**. En este caso se asignan diferentes pesos a las clases durante el entrenamiento del modelo, dando mayor importancia a la clase minoritaria. En últimas, se impone un peso mayor cuando se cometen errores en la clase minoritaria.

También se puede "lidiar" con este problema empleando métricas que no sean sensibles al desbalance de datos, como la precisión por clase o el F1-score. Otra opción es emplear algoritmos específicos para datos desbalanceados. Por ejemplo, existen algunos algoritmos de *machine learning* que están diseñados específicamente para manejar el desbalance de datos, como el *Random Oversampling Ensemble* (ROSE) o el *AdaBoost*.

Más adelante en este capítulo emplearemos la técnica de remuestreo *Down-sampling* para balancear nuestra muestra. Pero antes hablemos de los datos que emplearemos, el contexto y la pregunta de negocio.

10.2 La pregunta de negocio y los datos

Una gran superficie está preparando sus rebajas habituales de fin de año. El departamento de mercadeo lanzará una oferta similar a la lanzada el año anterior: la afiliación oro a un precio de $499 durante el evento de fin de año. La afiliación oro ofrece un 20% de descuento en todas las compras del año por un pago que regularmente es de $999. La oferta solo sería válida para los clientes actuales. Además la oferta implicaría una campaña telefónica en la que se llamaría a los clientes (Raza, 2023).

La gerencia cree que la mejor forma de reducir los costos de la campaña es crear un modelo que clasifique a los clientes que podrían adquirir la oferta y solo llamar a aquellos clientes con alta probabilidad de comprar la oferta.

Contamos con una base de datos con 22 variables que recopila los resultados de la campaña similar del año anterior. Los datos fueron provistos por Raza (2023) y están disponibles en el archivo `superstore_data.csv`[1].

Las variables con las que se cuentan son:

- `Response (target)`: 1 si el cliente aceptó la oferta en la última campaña, 0 en caso contrario.
- `ID`: ID único de cada cliente.
- `Year_Birth`: Edad del cliente.
- `Complain`: 1 si el cliente se quejó en los últimos 2 años.
- `Dt_Customer`: fecha de inscripción del cliente en la gran superficie.
- `Education`: nivel de estudios del cliente.
- `Marital`: estado civil del cliente.
- `Kidhome`: número de niños pequeños en el hogar del cliente.
- `Teenhome`: número de adolescentes en el hogar del cliente.
- `Income`: ingresos familiares anuales del cliente.
- `MntFishProducts`: gasto en productos de pescadería en los últimos 2 años.
- `MntMeatProducts`: gasto en productos cárnicos en los últimos 2 años
- `MntFruits`: gasto en productos de fruta en los últimos 2 años.
- `MntSweetProducts`: gasto en productos dulces en los últimos 2 años
- `MntWines`: gasto en productos vinícolas en los últimos 2 años.
- `MntGoldProds`: gasto en productos de oro en los últimos 2 años.
- `NumDealsPurchases`: número de compras realizadas con descuento.
- `NumCatalogPurchases`: número de compras realizadas por catálogo (compra de productos que se envían por correo).
- `NumStorePurchases`: número de compras realizadas directamente en tiendas.
- `NumWebPurchases`: número de compras realizadas a través del sitio web de la empresa.
- `NumWebVisitsMonth`: número de visitas al sitio web de la empresa en el último mes.
- `Recency`: número de días transcurridos desde la última compra.

[1] Los datos se pueden descargar de la página web del libro: http://www.icesi.edu.co/editorial/intro-clasificacion/.

10.3 Preprocesamiento de los datos

Carguemos primero los datos.

```
library(readr)
datos_originales <- read_csv("./datos/superstore_data.csv", col_types =
↪  cols(Education = col_factor(levels = c("2n Cycle",
    "Basic", "Graduation", "Master", "PhD")), Dt_Customer = col_datetime(format
    ↪  = "%m/%d/%Y"),
    Marital_Status = col_factor(levels = c("Absurd", "Alone", "Divorced",
    ↪  "Married",
        "Single", "Together", "Widow", "YOLO")), Complain = col_factor(levels =
        ↪  c("1",
        "0"))))
```

Nota que hay unas variables que por si solas no son útiles como `Year_Birth` (año de nacimiento) y `Dt_Customer` (fecha de inscripción del cliente en la gran superficie.), pero si podríamos emplearlas para calcular variables cuantitativas. Por ejemplo, creemos las variable edad del cliente (`edad_dias`), así como el número de días desde que el cliente está vinculado con el negocio (`Dias_cliente`). Supongamos que estamos el último día del año 2022. No es clara la fecha en que se suben los datos pero el archivo de Excel tiene fecha de creación el 2 de enero de 2023.

```
library(dplyr)
library(lubridate)

# Cambiar a formato fecha la variable Dt_Customer


hoy <- date(as.POSIXlt("12/31/2022", format = "%m/%d/%Y"))
# crear variable de días como cliente
datos_originales$Dias_cliente <- as.numeric(difftime(date(hoy),
↪  datos_originales$Dt_Customer,
    units = "days"))

# Crear variable de edad en años (aproximadamente)

datos_originales$edad_dias <- 2022 - datos_originales$Year_Birth
```

Además, noten que tenemos 24 valores perdidos en la variable `Income`. Esto nos generará problemas, como son tan pocos datos, borremos las filas que tengan algún dato perdido.

```
datos_originales <- datos_originales %>%
    na.omit()
```

Adicionalmente, separemos las variables cualitativas de las cuantitativas. En este caso también será necesario crear variables dummy para las variables cualitativas.

```
x_cuanti <- datos_originales %>%
    dplyr::select(Income, Kidhome, Teenhome, Recency, MntWines, MntFruits,
↪   MntMeatProducts,
        MntFishProducts, MntSweetProducts, MntGoldProds, NumDealsPurchases,
↪   NumWebPurchases,
        NumCatalogPurchases, NumStorePurchases, NumWebVisitsMonth,
↪   Dias_cliente,
        edad_dias)
# Estandarizar las variables cuanti

x_cuanti_estand <- x_cuanti %>%
    scale()

# Variables cualitativas
x_cuali <- datos_originales %>%
    dplyr::select(Education, Marital_Status, Complain)



# Crear variables dummy
library(fastDummies)
x_dummies <- dummy_cols(x_cuali, remove_most_frequent_dummy = TRUE,
↪   remove_selected_columns = TRUE)

# Crear variable objetivo
y <- relevel(as.factor(datos_originales$Response), ref = "1")

# Crear bases de datos 1 Variables cuali sin estandarizadas + dummies
datos_con_dummies <- cbind(y, x_cuanti, x_dummies)

# 2 Variables cuali estandarizadas + dummies
datos_estand_dummies <- cbind(y, x_cuanti_estand, x_dummies)

# 3 Variables solo cuali estandarizadas
datos_estand <- bind_cols(y, x_cuanti_estand)
names(datos_estand)[1] <- "y"

# 4 Variables solo cuali sin estandarizadas
datos_cuali <- as.data.frame(cbind(y, x_cuanti))

# 5 Variables cuali y cuantitativas
datos_estand_cuali <- cbind(y, x_cuanti_estand, x_cuali)
```

Noten que estamos trabajando con una muestra cuyo balance puede ser discutible.

```
table(y)/length(y) * 100
```

```
## y
##        1        0
## 15.02708 84.97292
```

Si bien en todo el libro hemos considerado una muestra balanceada aquella que tiene el 10% o menos de la muestra en la clase minoritaria, en este capítulo consideraremos que la muestra está balanceada (Ver Sección 10.1.1 para una discusión del tema).

Dada la aproximación que emplearemos para la validación cruzada y que emplearemos la "infraestructura" del paquete *caret* (Kuhn y Max, 2008), no será necesario crear la muestra de estimación y la muestra de evaluación.

10.4 Muestras desbalanceadas y validación cruzada con caret

Como se mencionó anteriormente, en este caso emplearemos la técnica de remuestreo *Down-sampling* que nos permita balancear nuestra muestra. Esta técnica se puede implementar fácilmente con el paquete *caret*. En la función **trainControl()** que ya conoces (Ver Sección 9.3) existe el argumento **sampling**. Este argumento tiene por defecto el valor de **"none"**, que corresponde a ningún remuestreo. Si **sampling = "down"**, se realizará un *Down-sampling*. Las otras opciones son **"up"** y **"smote"**.

Por otro lado, hasta aquí hemos realizado una validación cruzada de los modelos empleando el método de retención. Para emplear el método de *k-folds*, con $k = 5$ también podemos emplear la función **trainControl()**. Esta vez el argumento **method**, permite especificar el método de validación cruzada. Si **method = "cv"** se empleará el método *k-folds*, además será necesario emplear el argumento **number** para establecer el número de *folds*. Nota que si ponemos **number = 1** entonces estaremos empleando el método de retención. Si deseamos emplear el método de LOOCV entonces **method = "LOOCV"**.

10.5 Tuneo de los modelos en R

Es decir, para implementar el *Down-sampling* y la validacion de 5 iteraciones, emplearemos por lo menos los siguientes argumentos en la función **trainControl()**:

- **sampling = "down"**
- **method = "cv"**
- **number = 5**

Así mismo, para reducir el tiempo de computo, solo emplearemos el **Accuracy** como el criterio para seleccionar los mejores modelos al interior de los respectivos algoritmos. Es decir, emplearemos el argumento **metric = 'Accuracy'** en la función **train()**. Adicionalmente, para agilizar el proceso, omitiremos el modelo *Logit* de nuestos análisis. Tu puedes realizar los cálculos y encontrarás que este modelo tiene un mal comportamiento para estos datos. Procedamos a **tunear** los diferentes modelos.

10.5.1 Entrenamiento del modelo Naive Bayes

Noten que en este caso no es necesario *tunear* el modelo. Es decir, no es necesario hacer una búsqueda de parámetros. No obstante, podremos emplear la infraestructura de *caret* para realizar la validación cruzada con **k-folds** y el submuestreo.

Empecemos por establecer las condiciones bajo las cuales se realizará el ejercicio de "*tunning*". En este caso además estamos adicionando el argumento **savePredictions = final** para que se guarde la predicción en la muestra de evaluación de cada uno de los 5 *folds* para el modelo óptimo. Esto nos servirá para construir la tabla resumen que necesitaremos más adelante para comparar los resultados:

```
# Cargar la librería

library(caret)


# Fijar la forma como se realizará el tunninng
metodo_tune <- trainControl(
                    method = "cv", # k-fold
                    number = 5, # k =5 en k-fold
                    sampling = "down", # Down-sampling
                    savePredictions = "final" # Guardar todos los
                                              # y predichos fuera de
                                              # muestra
                    )
```

En este caso no se necesita hacer una búsqueda de grilla, por eso no tenemos que emplear la función **expand.grid()**.

Ahora, solo necesitamos entrenar el modelo *Naive Bayes* empleando la función **train()**. Para entrenar el modelo *Naive Bayes* debemos emplear **method = "nb"**. Esto empleará la correspondiente función del paquete *klaR*. Sino cuentas con este modelo, lo debes instalar antes.

En este caso la línea de código será:

```
set.seed(123)

## empezar el tunning para el modelo Naive Bayes
modelo_nb <- train(form = y ~ ., data = datos_cuali, method = "nb", trControl =
↪  metodo_tune,
    metric = "Accuracy")
```

Noten que idealmente deberíamos emplear la base de datos con los datos cuantitativos y las dummies. No obstante, dada las pocas observaciones de algunas clases (como por ejemplo "YOLO" en `Marital`) esto genera problemas de poca o nula varianza en algunos de los *folds*. Por esto optamos por emplear solo las variables cuantitativas. Este es un problema de esta base y no necesariamente será igual en todas las bases

de datos.

Veamos los resultados:

```
modelo_nb
```

```
## Naive Bayes
##
## 2216 samples
##   17 predictor
##    2 classes: '1', '0'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1772, 1773, 1773, 1774, 1772
## Addtional sampling using down-sampling
##
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.6827683  0.1920558
##    TRUE      0.6588630  0.1780516
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = FALSE and adjust
##  = 1.
```

En este caso, las métricas (promedio) para los *5-folds* los podemos encontrar empleando la función **confusionMatrix()**.

```
confusionMatrix(modelo_nb)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    1    0
##          1  9.0 25.7
##          0  6.0 59.3
##
##  Accuracy (average) : 0.6828
```

10.5.2 Entrenamiento del modelo kNN

Sigamos empleando la infraestructura del paquete *caret* para entrenar el modelo de árboles de decisión. Inicialmente, definamos cómo se realizará el ejercicio de *tunning*.

```
# Fijar la forma como se realizará el tunninng
metodo_tune <- trainControl(
                    method = "cv", # k-fold
                    number = 5, # k =5 en k-fold
                     search = "grid",
                    sampling = "down", # Down-sampling
                     savePredictions = "final" # Guardar todos los
                                               # y predichos fuera de
                                               # muestra
                    )
```

Recodemos que **kNN** no permite usar variables cualitativas y las cualitativas deberían estar estandarizas. Así emplearemos el objeto `datos_estand`.

En este caso, el parámetro que debemos "*tunear*" es k. Emplearemos una grilla de búsqueda inusualmente alta, porque al empezar con una grilla pequeña, el k óptimo estaba muy cerca al valor superior de la grilla (inténtalo). De esta manera, se "jugo" iterativamente con el límite superior de la grilla, hasta encontrar que la escogencia del límite superior de la grilla no tenía un efecto sobre el óptimo seleccionado.

Recordemos que cuando se entrena un modelo de *Machine Learning*, se utiliza un conjunto de datos que se supone que es una muestra de la distribución estadística real que se desea modelar. Esto significa, que el conjunto de datos no refleja totalmente la distribución real y contendrá anomalías, excepciones o cierta aleatoriedad. Un modelo sobreajustado (*overfit*) se ceñiría demasiado a las particularidades de este conjunto de datos y sería demasiado "variable", en lugar de aprender patrones estadísticos que suavizarían estas particularidades y acercarían el modelo a la distribución estadística real.

En el caso de **kNN**, k controla el tamaño del vecindario utilizado para modelar las propiedades estadísticas locales. Un valor muy pequeño de k hace que el modelo sea más sensible a las anomalías y excepciones locales, dando demasiado peso a estos puntos concretos. Por el contrario, un valor de k demasiado grande haría que el modelo ignorara la estructura local de la distribución que intenta aprender, y produciría un modelo infraajustado (*underfit*). Cómo se discutió en el Capítulo 6 es importante encontrar el k óptimo y lo mejor sería una búsqueda de grilla.

Para establecer el máximo k en nuestra búsqueda de grilla seguiremos a Dasarathy (1991) (Ver Capítulo 6). En nuestro caso $P = 333$ y $N = 1883$, olvidando que emplearemos la validación cruzada de *5-folds*, el k máximo para nuestra búsqueda de grilla sería 666. Como tendremos una validación cruzada de *5-folds* tendremos aproximadamente 80% las observaciones en la muestra de entrenamiento. Así redondeando emplearemos como límite máximo $k = 500$. Así tendremos:

```
# Establecer la grilla de búsqueda para el costo
grilla_busqueda <- expand.grid(k = seq(3, 500, by = 1))
```

Y empleemos ahora la función **train()**.

```
modelo_kNN <- train(form = y ~ ., data = datos_estand, method = "knn",
↪  trControl = metodo_tune,
    tuneGrid = grilla_busqueda, metric = "Accuracy")
```

Veamos el valor de k que maximiza el **accuracy**

```
modelo_kNN$bestTune
```

```
##       k
## 469 471
```

Y la matriz de confusión promedio de los *5-folds* es:

```
confusionMatrix(modelo_kNN)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    1    0
##          1  6.1 11.5
##          0  8.9 73.5
##
##  Accuracy (average) : 0.796
```

10.5.3 Entrenamiento del modelo árboles de decisión

Sigamos empleando la infraestructura del paquete *caret* para entrenar el modelo de árboles de decisión.

En este caso no es necesario definir nuevamente cómo se realizará el ejercicio de *tunning*, pues es igual al caso anterior. En este caso el parámetro *tunear* es **cp**.

```
# Establecer la grilla de búsqueda para el costo

grilla_busqueda <- expand.grid(cp = c(0, 0.1, 1, 10, 50, 70))
```

Y ahora entrenemos el modelo empleando **method = "rpart"** para invocar la función **rpart()** del paquete *rpart* que estudiamos anteriormente y **parms = list(split = "gini")** para definir el coeficiente de Gini como el criterio para generar los *splits*. Recuerden que en este caso podemos emplear las variables cualitativas estandarizadas y las variables dummy.

```
## arboles de decisión usando coeficiente de entropía
modelo_DT <- train(y ~ ., data = datos_estand_cuali, method = "rpart", metric =
↪  "Accuracy",
    trControl = metodo_tune, tuneGrid = grilla_busqueda, parms = list(split =
    ↪  "gini"))
```

Veamos los resultados.

```
modelo_DT$bestTune
```

```
##   cp
## 1  0
```

En este caso, las métricas (promedio) para los *5-folds* los podemos encontrar empleando la función **confusionMatrix()**.

```
confusionMatrix(modelo_DT)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    1    0
##          1 11.0 24.5
##          0  4.0 60.4
##
##  Accuracy (average) : 0.7144
```

10.5.4 Entrenamiento del modelo Random Forest

Emplearemos la misma definición del ejercicio de *tunning* que antes (ver objeto `metodo_tune`). Recordemos que en este caso el hiperparámetro que se debe *tunear* es **mtry** (El número de variables que se remuestrean aleatoriamente para ser muestreado en cada *split*)[2].

```
grilla_busqueda <- expand.grid(.mtry = seq(1, sqrt(ncol(datos_estand_cuali) -
↪  1)))
```

```
set.seed(123)
modelo_rf <- train(y ~ ., data = datos_estand_cuali, method = "rf", trControl =
↪  metodo_tune,
    tuneGrid = grilla_busqueda, metric = "Accuracy")
```

Veamos los resultados.

[2]Recuerda que por defecto el número de ramas (`ntree`) que se crecerán después de cada división temporal.

```
modelo_rf$bestTune
```

```
##   mtry
## 3    3
```

En este caso, las métricas (promedio) para los *5-folds* los podemos encontrar empleando la función **confusionMatrix()**.

```
confusionMatrix(modelo_rf)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    1    0
##          1 12.1 20.6
##          0  2.9 64.4
##
##  Accuracy (average) : 0.7644
```

10.5.5 Entrenamiento del modelo SMV con kernel lineal

Emplearemos el mismo `metodo_tune` y en este caso tenemos:

```
# Establecer la grilla de búsqueda para el costo
grilla_busqueda <- expand.grid(C = c(0.01, 0.1, 1, 10, 50))

set.seed(123)

## empezar el tunning para el modelo svm con kernel lineal
modelo_svm_lineal <- train(form = y ~ ., data = datos_estand_cuali, method =
↪  "svmLinear",
    trControl = metodo_tune, metric = "Accuracy", tuneGrid = grilla_busqueda)
```

Veamos los resultados.

```
modelo_svm_lineal$bestTune
```

```
##   C
## 3 1
```

En este caso, las métricas (promedio) para los *5-folds* los podemos encontrar empleando la función **confusionMatrix()**.

```
confusionMatrix(modelo_svm_lineal)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
```

```
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    1    0
##          1 11.7 20.9
##          0  3.3 64.0
##
##  Accuracy (average) : 0.7577
```

10.5.6 Entrenamiento del modelo SVM con kernel polinómico

De manera similar, el código para el modelo **SMV** con *kernel* polinómico será:

```
# Establecer la grilla de búsqueda para el modelo SVM kernel polimonial
grilla_busqueda <- expand.grid(C = c(0.01, 0.1, 1, 10), degree = c(2, 3), scale
↪  = 0.01)

# Tunear el modelo SVM kernel polimonial

modelo_svm_pol <- train(y ~ ., data = datos_estand_cuali, method = "svmPoly",
↪  trControl = metodo_tune,
    metric = "Accuracy", tuneGrid = grilla_busqueda)
```

Veamos los resultados.

```
modelo_svm_pol$bestTune
```

```
##   degree scale    C
## 2      3  0.01 0.01
```

En este caso, las métricas (promedio) para los *5-folds* los podemos encontrar empleando la función **confusionMatrix()**.

```
confusionMatrix(modelo_svm_pol)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    1    0
##          1  4.3  9.1
##          0 10.7 75.9
##
##  Accuracy (average) : 0.8019
```

10.5.7 Entrenamiento del modelo SVM con kernel RBF

Ya será evidente para tí que el código para el modelo **SMV** con *kernel* **RBF** será:

```
# Establecer la grilla de búsqueda para el modelo SVM kernel RBF

grilla_busqueda <- expand.grid(C = c(0.01, 0.1, 1, 10), sigma = c(0.01, 0.1,
↪  1))

modelo_svm_rbf <- train(y ~ ., data = datos_estand_cuali, method = "svmRadial",
↪  trControl = metodo_tune,
    metric = "Accuracy", tuneGrid = grilla_busqueda)
```

Veamos los resultados.

```
modelo_svm_rbf$bestTune
```

```
##   sigma C
## 9     1 1
```

En este caso, las métricas (promedio) para los *5-folds* los podemos encontrar empleando la función **confusionMatrix()**.

```
confusionMatrix(modelo_svm_rbf)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    1    0
##          1  3.0  7.2
##          0 12.0 77.8
##
##  Accuracy (average) : 0.8078
```

10.6 Comparación de los modelos

Ahora que tenemos los respectivos objetos con los mejores modelos tuneados, podemos proceder a construir la tabla resumen. Nota que los valores predichos para cada *fold* del mejor modelo se encuentra en el *slot* **pred** de los respectivos objetos. El siguiente código produce los resultados que se presentan en el Cuadro 10.1.

```
library(caret)
# métricas Naive Bayes
metricas_nb <- data.frame()
acc_nb <- data.frame()

for (i in unique(modelo_nb$pred$Resample)) {
    base_t <- modelo_nb$pred %>%
        filter(Resample == i)
```

```
    m <- confusionMatrix(base_t$pred, base_t$obs)
    metricas_nb <- rbind(metricas_nb, m$byClass)
    acc_nb <- rbind(acc_nb, m$overall[1])
}
names(metricas_nb) <- names(m$byClass)
names(acc_nb) <- names(m$overall[1])
metricas_nb <- bind_cols(metricas_nb, acc_nb)
metricas_nb$Modelo <- "Naive Bayes"

# métricas kNN
metricas_kNN <- data.frame()
acc_kNN <- data.frame()
for (i in unique(modelo_kNN$pred$Resample)) {
    base_t <- modelo_kNN$pred %>%
        filter(Resample == i)

    m <- confusionMatrix(base_t$pred, base_t$obs)
    metricas_kNN <- rbind(metricas_kNN, m$byClass)
    acc_kNN <- rbind(acc_kNN, m$overall[1])
}
names(metricas_kNN) <- names(m$byClass)
names(acc_kNN) <- names(m$overall[1])
metricas_kNN <- bind_cols(metricas_kNN, acc_kNN)
metricas_kNN$Modelo <- "kNN"

# métricas DT_gini_500obs
metricas_DT <- data.frame()
acc_DT <- data.frame()
for (i in unique(modelo_DT$pred$Resample)) {
    base_t <- modelo_DT$pred %>%
        filter(Resample == i)

    m <- confusionMatrix(base_t$pred, base_t$obs)
    metricas_DT <- rbind(metricas_DT, m$byClass)
    acc_DT <- rbind(acc_DT, m$overall[1])
}
names(metricas_DT) <- names(m$byClass)
names(acc_DT) <- names(m$overall[1])
metricas_DT <- bind_cols(metricas_DT, acc_DT)
metricas_DT$Modelo <- "DT gini 500obs"

# métricas Randomo Forest
metricas_rf <- data.frame()
acc_rf <- data.frame()
```

```
for (i in unique(modelo_rf$pred$Resample)) {
    base_t <- modelo_rf$pred %>%
        filter(Resample == i)

    m <- confusionMatrix(base_t$pred, base_t$obs)
    metricas_rf <- rbind(metricas_rf, m$byClass)
    acc_rf <- rbind(acc_rf, m$overall[1])
}
names(metricas_rf) <- names(m$byClass)
names(acc_rf) <- names(m$overall[1])
metricas_rf <- bind_cols(metricas_rf, acc_rf)
metricas_rf$Modelo <- "Random Forest"

# métricas SVM lineal
metricas_svm_lineal <- data.frame()
acc_svm_lineal <- data.frame()
for (i in unique(modelo_svm_lineal$pred$Resample)) {
    base_t <- modelo_svm_lineal$pred %>%
        filter(Resample == i)

    m <- confusionMatrix(base_t$pred, base_t$obs)
    metricas_svm_lineal <- rbind(metricas_svm_lineal, m$byClass)
    acc_svm_lineal <- rbind(acc_svm_lineal, m$overall[1])
}
names(metricas_svm_lineal) <- names(m$byClass)
names(acc_svm_lineal) <- names(m$overall[1])
metricas_svm_lineal <- bind_cols(metricas_svm_lineal, acc_svm_lineal)
metricas_svm_lineal$Modelo <- "SVM lineal"

# métricas SVM polinomial
metricas_svm_pol <- data.frame()
acc_svm_pol <- data.frame()
for (i in unique(modelo_svm_pol$pred$Resample)) {
    base_t <- modelo_svm_pol$pred %>%
        filter(Resample == i)

    m <- confusionMatrix(base_t$pred, base_t$obs)
    metricas_svm_pol <- rbind(metricas_svm_pol, m$byClass)
    acc_svm_pol <- rbind(acc_svm_pol, m$overall[1])
}
names(metricas_svm_pol) <- names(m$byClass)
names(acc_svm_pol) <- names(m$overall[1])
metricas_svm_pol <- bind_cols(metricas_svm_pol, acc_svm_pol)
metricas_svm_pol$Modelo <- "SVM pol"
```

```
# métricas SVM RBF
metricas_svm_rbf <- data.frame()
acc_svm_rbf <- data.frame()
for (i in unique(modelo_svm_rbf$pred$Resample)) {
    base_t <- modelo_svm_rbf$pred %>%
        filter(Resample == i)

    m <- confusionMatrix(base_t$pred, base_t$obs)
    metricas_svm_rbf <- rbind(metricas_svm_rbf, m$byClass)
    acc_svm_rbf <- rbind(acc_svm_rbf, m$overall[1])
}
names(metricas_svm_rbf) <- names(m$byClass)
names(acc_svm_rbf) <- names(m$overall[1])
metricas_svm_rbf <- bind_cols(metricas_svm_rbf, acc_svm_rbf)
metricas_svm_rbf$Modelo <- "SVM RBF"

# unir todas las métricas
tabla_1 <- bind_rows(metricas_nb, metricas_kNN, metricas_DT, metricas_rf,
↪  metricas_svm_lineal,
    metricas_svm_pol, metricas_svm_rbf)
# Extraer las métricas que queremos
tabla <- tabla_1 %>%
    group_by(Modelo) %>%
    summarise(`Exactitud (ACC)` = mean(Accuracy), `Sensibiliad (TPR) (Recall)`
    ↪  = mean(Sensitivity),
        `Especificidad (TNR)` = mean(Specificity), `Precisión (PPV)` =
        ↪  mean(Precision),
        F1 = mean(F1)) %>%
    arrange(desc(`Precisión (PPV)`))
```

La mejor métrica para responder la pregunta de negocio parece ser la precisión[3]. Por eso el mejor modelo sería el **SVM pol**.

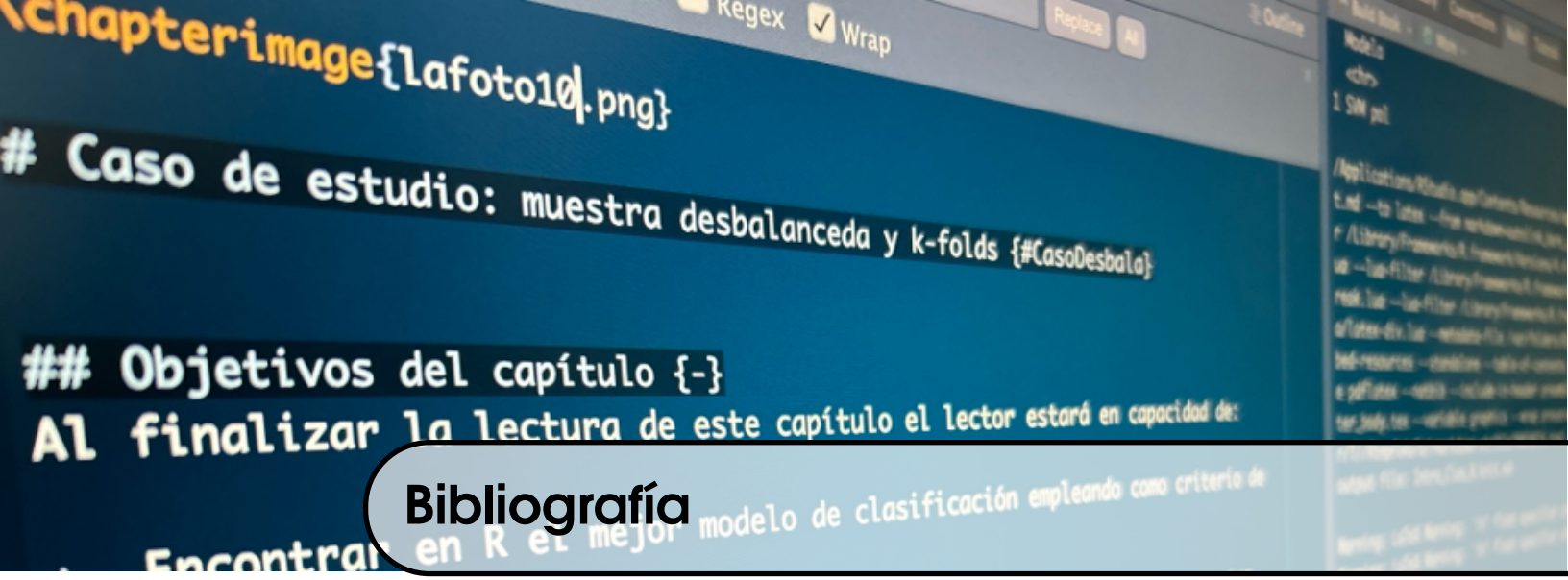
Una vez establecido que éste es el mejor modelo, se emplearía para predecir sobre la base de datos de los actuales clientes. De esta manera el modelo debería proveer un listado de los clientes actuales que se predicen como "1". Es decir, aquellos que comprarían la afiliación oro. Es decir, el listado de clientes clasificados como "1" debería ser el producto que se le entrega al departamento de mercadeo. El departamento de mercadeo llamaría a todos los clientes que hagan parte de este listado.

[3]Esta métrica debería discutirse con los tomadores de decisiones que emplearán el modelo. Si la organización cuenta con un *analytics translator*, esa será la persona indicada para discutir si esa métrica si es la adecuada.

Cuadro 10.1. Métricas de comparación para todos los modelos de clasificación (organizadas por precisión)

Modelo	Exactitud (ACC)	Sensibiliad (TPR) (Recall)	Especificidad (TNR)	Precisión (PPV)	F1
SVM pol	0.802	0.287	0.893	0.457	0.272
Random Forest	0.764	0.805	0.757	0.371	0.507
SVM lineal	0.758	0.781	0.754	0.360	0.492
kNN	0.796	0.409	0.865	0.346	0.374
DT gini 500obs	0.714	0.733	0.711	0.311	0.436
SVM RBF	0.808	0.201	0.915	0.289	0.235
Naive Bayes	0.683	0.598	0.698	0.259	0.361

Fuente: elaboración propia.



Bibliografía

Alonso, J. C. (2022). *Empezando a transformar bases de datos con R y dplyr*. Universidad Icesi.

Alonso, J. C. (2024). *Introducción al Modelo Clásico de Regresión para Científico de Datos en R*. Universidad Icesi.

Alonso, J. C. y Arboleda, A. M. (2024). *Introducción al Análisis de Canastas de Compra para analytics translators y científicos de datos (empleando R)*. Universidad Icesi.

Alonso, J. C. y Arboleda, A. M. (2025). *Introducción al Análisis de Canastas de Compra para analytics translators y científicos de datos (empleando R)*. Universidad Icesi.

Alonso, J. C. y Largo, M. F. (2023). *Empezando a visualizar datos con R y ggplot2*. Universidad Icesi, 2. edition.

Alonso, J. C. y Ocampo, M. P. (2022). *Empezando a usaR: Una guía paso a paso*. Universidad Icesi.

Aquino, J. (2023). *descr: Descriptive Statistics*. R package version 1.1.8.

Batista, G. E., Prati, R. C., y Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, 6(1):20–29.

Breiman, L., Friedman, J., Olshen, R., y Stone, C. (1984). Classification and regression trees. wadsworth & brooks. *Cole Statistics/Probability Series*.

Chawla, N. V., Bowyer, K. W., Hall, L. O., y Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357.

Cox, D. R. y Snell, E. J. (1989). *Analysis of binary data*, volume 32. CRC press.

Dasarathy, B. V. (1991). Nearest neighbor (nn) norms: Nn pattern classification techniques. *IEEE Computer Society Tutorial*.

Fernihough, A. (2019). *mfx: Marginal Effects, Odds Ratios and Incidence Rate Ratios for GLMs*. R package version 1.2-2.

Firke, S. (2023). *janitor: Simple Tools for Examining and Cleaning Dirty Data*. R package version 2.2.0.

García, S., Fernández, A., Luengo, J., y Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information sciences*, 180(10):2044–2064.

He, H. y Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284.

Kaplan, J. (2023). *fastDummies: Fast Creation of Dummy (Binary) Columns and Rows from Categorical Variables*. R package version 1.7.3.

Khan, M. R. A. y Brandenburger, T. (2020). *ROCit: Performance Assessment of Binary Classifier with Visualization*. R package version 2.1.1.

Kuhn y Max (2008). Building predictive models in r using the caret package. *Journal of Statistical Software*, 28(5):1–26.

Lall, U. y Sharma, A. (1996). A nearest neighbor bootstrap for resampling hydrologic time series. *Water Resources Research*, 32(3):679–693.

Leeper, T. J. (2021). *margins: Marginal Effects for Model Objects*. R package version 0.3.26.

Liaw, A. y Wiener, M. (2002). Classification and regression by randomforest. *R News*, 2(3):18–22.

Majka, M. (2019). *naivebayes: High Performance Implementation of the Naive Bayes Algorithm in R*. R package version 0.9.7.

McFadden, D. (1972). Conditional logit analysis of qualitative choice behavior. Technical report.

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., y Leisch, F. (2023). *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.7-13.

Milborrow, S. (2022). *rpart.plot: Plot 'rpart' Models: An Enhanced Version of 'plot.rpart'*. R package version 3.1.1.

Moro, S., Cortez, P., y Rita, P. (2014). A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31.

Nagelkerke, N. J. et al. (1991). A note on a general definition of the coefficient of determination. *biometrika*, 78(3):691–692.

Ng, A. (2014). Machine learning lecture notes.

R Core Team (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Raza, A. (2023). Superstore marketing campaign dataset. Kaggle.

Therneau, T. y Atkinson, B. (2022). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1.19.

Venables, W. N. y Ripley, B. D. (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0.

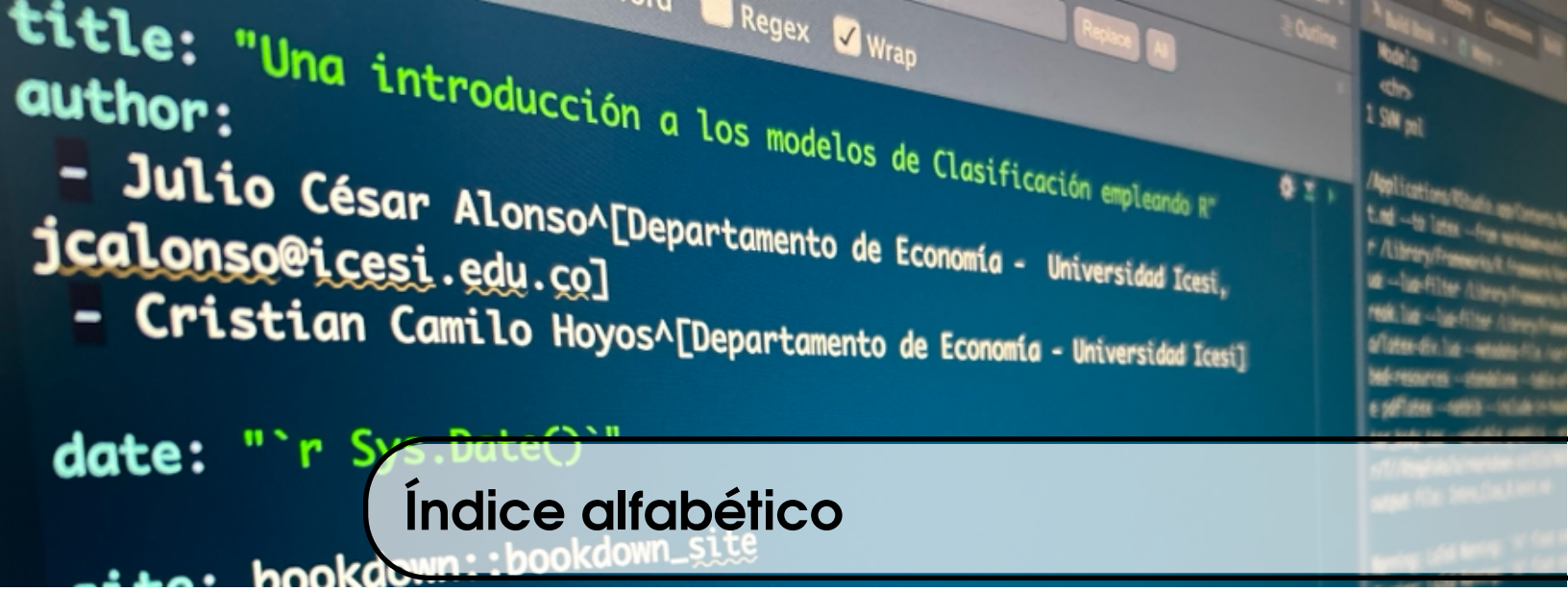
Wang, S.-j., Mathew, A., Chen, Y., Xi, L.-f., Ma, L., y Lee, J. (2009). Empirical analysis of support vector machine ensemble classifiers. *Expert Systems with applications*, 36(3):6466–6476.

Weiss, G. M. y Provost, F. (2003). Learning when training data are costly: The effect of class distribution on tree induction. *Journal of artificial intelligence research*, 19:315–354.

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.

Wickham, H., François, R., Henry, L., y Müller, K. (2021). *dplyr: A Grammar of Data Manipulation*. R package version 1.0.7.

Youden, W. J. (1950). Index for rating diagnostic tests. *Cancer*, 3(1):32–35.



Índice alfabético

Universidad Icesi

Editorial Universidad Icesi

Este libro presenta una introducción a los modelos estadísticos y de aprendizaje de máquina que permiten realizar la tarea de clasificación. La discusión de los diferentes capítulos está dirigida a personas que están empezando su formación de científico de datos. Esta obra recoge nuestra experiencia trabajando con R y los modelos de clasificación para resolver problemas con datos desde el Centro de Investigación en Economía y Finanzas (Cienfi) de la Universidad Icesi, y transformar estos datos en conclusiones que faciliten la toma de decisiones en organizaciones privadas y públicas.

Herramientas del BIG DATA y analytics