Datos desafiantes – Demasiados, muy pocos, muy complejos, parte II

Ejemplo - Remapeo de datos categóricos dispersos

Como vimos en temas anteriores, al añadir una característica categórica a un conjunto de datos, esta suele transformarse en un conjunto de variables binarias igual al número de niveles de la característica original mediante codificación ficticia o one-hot.

Por ejemplo, si hay 40,000 códigos postales en Estados Unidos, el algoritmo de aprendizaje automático tendría 40,000 predictores binarios para esta característica. Esto se denomina mapeo uno de *n* (*one-of-n-mapping*) porque solo una de las 40,000 características tendría el valor 1, mientras que el resto tendría valores 0; un caso de crecimiento extremo en dimensionalidad y dispersión.

Para aumentar la densidad de un mapeo uno de *n*, se puede utilizar un mapeo *m* de *n* (*m-of-n mapping*), que reduce las *n* variables binarias a un conjunto más pequeño de *m* variables. Por ejemplo, con los códigos postales, en lugar de crear una característica de 40,000 niveles con un nivel por código postal, se podría optar por mapear en 100 niveles utilizando los dos primeros dígitos del código postal, del 00 al 99. De igual manera, si incluir una característica binaria para cada uno de los 200 países del mundo generara demasiada dispersión, se podría mapear los países a un conjunto más pequeño de continentes, como Europa, Norteamérica y Asia.

Al crear un mapeo *m*-de-*n*, es mejor que las agrupaciones representen una característica subyacente compartida, pero también es posible utilizar otros enfoques. El conocimiento del dominio puede ser útil para crear un remapeo que refleje las características compartidas de las unidades más granulares. Si no se cuenta con experiencia en el dominio, los siguientes métodos pueden ser apropiados:

- Dejar las categorías más grandes como están y agrupar solo las categorías con un número reducido de observaciones. Por ejemplo, los códigos postales de zonas urbanas densas podrían incluirse directamente, pero los códigos postales rurales dispersos podrían agruparse en áreas geográficas más grandes.
- Examinar el impacto de las categorías en la variable objetivo creando una tabla cruzada de dos vías (*two-way*) o calculando el resultado promedio por nivel y niveles de grupo que tengan un impacto similar en la variable de respuesta. Por ejemplo, si

ciertos códigos postales tienen mayor probabilidad de impago de un préstamo, cree una nueva categoría compuesta por estos códigos postales.

- Como una variante más sofisticada del método anterior, también se puede construir un modelo simple de aprendizaje automático que prediga el objetivo utilizando la característica altamente dimensional y luego agrupe los niveles de la característica que tengan una relación similar con el objetivo o con otros predictores. Métodos simples como la regresión y los árboles de decisión serían ideales para este enfoque.

Una vez elegida la estrategia de reasignación, se pueden encontrar funciones útiles para recodificar variables categóricas en el paquete forcats (https://forcats.tidyverse.org), que forma parte del conjunto básico de paquetes que componen tidyverse. El paquete incluye opciones para recodificar automáticamente variables categóricas con niveles dispersos o para recodificar manualmente si se desea un enfoque más guiado.

Examinaremos un par de enfoques para la reasignación utilizando el conjunto de datos del Titanic y los títulos de los pasajeros. Dado que el paquete forcats está incluido en el paquete base tidyverse, puede cargarse con todo el conjunto o de forma independiente mediante el comando library(forcats). Comenzaremos cargando tidyverse, leyendo el conjunto de datos del Titanic y, a continuación, examinando los niveles de la función de título:

```
> library(tidyverse)
> library(forcats)
> titanic_train <- read_csv("titanic_train.csv") |>
+   mutate(Title = str_extract(Name, ", [A-z]+\\.")) |>
+   mutate(Title = str_replace_all(Title, "[, \\.]", ""))
> table(titanic_train$Title, useNA = "ifany")
```

```
  Capt      Col      Don       Dr Jonkheer     Lady    Major   Master     Miss     Mlle      Mme       Mr      Mrs
     1        2        1        7        1        1        2       40      182        2        1      517      125
    Ms      Rev      Sir     <NA>
     1        6        1        1
```

Anteriormente usamos la función recode() de R base para combinar las variantes de Miss, como Ms, Mlle y Mme, en un solo grupo. El paquete forcats incluye la función fct_collapse(), más práctica para entidades categóricas con un gran número de niveles.
La usaremos aquí para crear una asignación *m* de *n* que genere grupos basándose en el conocimiento del significado real de los títulos. Cabe destacar que varias de las nuevas categorías son asignaciones biunívocas de las categorías anteriores, pero al incluir un vector de etiquetas, podemos asignar varios de los niveles antiguos a un único nivel nuevo, como se indica a continuación:

```
> titanic_train <- titanic_train |>
+   mutate(TitleGroup = fct_collapse(Title,
```

```
+    Mr = "Mr",
+    Mrs = "Mrs",
+    Master = "Master",
+    Miss = c("Miss", "Mlle", "Mme", "Ms"),
+    Noble = c("Don", "Sir", "Jonkheer", "Lady"),
+    Military = c("Capt", "Col", "Major"),
+    Doctor = "Dr",
+    Clergy = "Rev",
+    other_level = "Other")
+  ) |>
+  mutate(TitleGroup = fct_na_value_to_level(TitleGroup,
+    level = "Unknown"))
```

Al examinar la nueva categorización, observamos que las 17 categorías originales se han reducido a 9:

```
> table(titanic_train$TitleGroup)
Military    Noble   Doctor   Master     Miss       Mr      Mrs   Clergy  Unknown
       5        4        7       40      186      517      125        6        1
```

Si tuviéramos un conjunto de niveles mucho mayor, o si no supiéramos cómo agrupar las categorías, podemos dejar las categorías grandes como están y agrupar los niveles con algunos ejemplos. El paquete forcats incluye una función sencilla para examinar los niveles de nuestra entidad. Aunque esto también se puede realizar con funciones básicas de R, la función fct_count() proporciona una lista ordenada de los niveles de características y sus proporciones respecto al total general:

```
> fct_count(titanic_train$Title, sort = TRUE, prop = TRUE)
```

```
# A tibble: 17 × 3
   f            n       p
   <fct>    <int>   <dbl>
 1 Mr         517 0.580
 2 Miss       182 0.204
 3 Mrs        125 0.140
 4 Master      40 0.0449
 5 Dr           7 0.00786
 6 Rev          6 0.00673
 7 Col          2 0.00224
 8 Major        2 0.00224
 9 Mlle         2 0.00224
10 Capt         1 0.00112
11 Don          1 0.00112
12 Jonkheer     1 0.00112
13 Lady         1 0.00112
14 Mme          1 0.00112
15 Ms           1 0.00112
16 Sir          1 0.00112
17 <NA>         1 0.00112
```

Esta salida puede informar agrupaciones basadas en un número mínimo o una proporción mínima de observaciones. El paquete forcats incluye un conjunto de funciones fct_lump() para facilitar este proceso de agrupar los niveles de factores en un grupo "otros". Por ejemplo, podríamos tomar los tres niveles superiores y tratar todo lo demás como “otros”:

```
> table(fct_lump_n(titanic_train$Title, n = 3))

 Miss    Mr   Mrs   Other
  182   517   125     66
```

Alternativamente, podemos agrupar todos los niveles con menos del uno por ciento de las observaciones:

```
> table(fct_lump_prop(titanic_train$Title, prop = 0.01))

Master   Miss     Mr    Mrs  Other
   40    182     517    125     26
```

Por último, podríamos optar por agrupar todos los niveles con menos de cinco observaciones:

```
> table(fct_lump_min(titanic_train$Title, min = 5))

   Dr  Master   Miss     Mr    Mrs    Rev   Other
    7      40    182    517    125      6      13
```

La elección de cuál de estas tres funciones utilizar, así como el valor del parámetro adecuado, dependerá del conjunto de datos utilizado y del número de niveles deseado para la asignación *m* de *n*.

Ejemplo – Agrupamiento (clasificación, *binning*) de datos numéricos dispersos

Si bien muchos métodos de aprendizaje automático procesan datos numéricos sin problemas, algunos enfoques, como los árboles de decisión, tienden a presentar dificultades con estos datos, especialmente cuando presentan características de dispersión. Una solución común a este problema se denomina discretización, que convierte un rango de números en un número menor de categorías discretas llamadas bins. Encontramos este método previamente en el tema, Aprendizaje Probabilístico: Clasificación con Naive Bayes, al discretizar los datos numéricos para que funcionaran con el algoritmo Naive Bayes. Aquí, aplicaremos un enfoque similar, utilizando métodos modernos de tidyverse, para reducir la dimensionalidad del rango

numérico y así abordar la tendencia de algunos métodos a sobreajustarse o subajustarse a datos numéricos dispersos.

Como ocurre con muchos enfoques de aprendizaje automático, lo ideal sería aplicar conocimientos especializados para determinar los puntos de corte para discretizar un rango numérico. Por ejemplo, en un rango de valores de edad, podrían existir puntos de corte significativos entre los grupos de edad bien establecidos de infancia, adultez y vejez, para reflejar el impacto de estos valores en la vida real.
De igual forma, se pueden crear intervalos para niveles salariales como clase baja, media y alta.

Ante la falta de conocimiento práctico de categorías importantes, suele ser recomendable utilizar puntos de corte que reflejen percentiles naturales de datos o incrementos intuitivos de valores. Esto puede implicar dividir un rango de números mediante estrategias como:

- Crear grupos basados en terciles, cuartiles, quintiles, deciles o percentiles que contengan proporciones iguales de ejemplos (33 %, 25 %, 20 %, 10 % o 1 %).
- Utilizar puntos de corte habituales para el rango subyacente de valores, como agrupar los valores de tiempo por horas, medias horas o cuartos de hora; agrupar los valores de la escala de 0 a 100 por cincos, decenas o veinticinco; o agrupar rangos numéricos amplios, como los ingresos, en múltiplos grandes de 10 o 25.
- Aplicar el concepto de escala logarítmica a datos sesgados, de modo que los intervalos sean proporcionalmente más amplios para la parte sesgada de los datos donde los valores son más dispersos; por ejemplo, los ingresos podrían agruparse en grupos de 0 a 10,000, seguidos de 10,000 a 100,000, luego de 100,000 a 1,000,000, y finalmente de 1,000.000 o más.

Para ilustrar estos enfoques, aplicaremos técnicas de discretización a los valores de tarifa (*fare*) en el conjunto de datos del Titanic utilizado anteriormente. Las funciones head() y summary() ilustran que los valores son muy granulares y dispersos en el extremo superior debido a su marcada asimetría a la derecha:

```
> head(titanic_train$Fare)
[1]  7.2500  71.2833  7.9250  53.1000  8.0500  8.4583
> summary(titanic_train$Fare)
   Min.  1st Qu.  Median    Mean   3rd Qu.    Max.
   0.00    7.91     14.45    32.20    31.00   512.33
```

Supongamos que nos interesa principalmente la diferencia entre pasajeros de primera clase y otros pasajeros, y que asumimos que el 25% superior de las tarifas corresponden a billetes de primera clase. Podríamos crear fácilmente una característica binaria utilizando la función

if_else() de tidyverse de la siguiente manera. Si la tarifa tiene un valor de al menos £31, que corresponde al tercer cuartil, asumiremos que es de primera clase y asignaremos el valor 1 a la característica fare_firstclass (codificada en binario); de lo contrario, recibirá el valor 0. El parámetro missing indica a la función que asigne el valor 0 si la tarifa falta, asumiendo que es muy improbable que las tarifas de primera clase sean desconocidas:

```
> titanic_train <- titanic_train |> mutate(
+   fare_firstclass = if_else(Fare >= 31, 1, 0, missing = 0)
+ )
```

Esto reduce una característica con casi 250 valores distintos a una nueva característica con solo dos:

```
> table(titanic_train$fare_firstclass)
  0   1
666 225
```

Aunque este fue un ejemplo muy simple, es un primer paso hacia estrategias de binning más complejas.
La función if_else(), aunque simple en este caso, sería difícil de usar para crear una nueva característica con más de dos niveles. Esto requeriría anidar funciones if_else() entre sí, lo que rápidamente se vuelve difícil de mantener. En su lugar, una función de tidyverse llamada case_when() permite la construcción de una serie de comprobaciones más complejas para determinar el resultado.

En el código siguiente, los datos de tarifas se agrupan en tres niveles que corresponden aproximadamente a los niveles de tarifa de primera, segunda y tercera clase. La sentencia case_when() se evalúa como una serie de sentencias if-else ordenadas. La primera sentencia comprueba si la tarifa es al menos 31 y asigna a estos ejemplos la categoría de primera clase.

La segunda puede interpretarse como una sentencia else-if; es decir, si la primera sentencia no es verdadera (el "else"), comprobamos si la tarifa es al menos 15 y, si es verdadera, asignamos el nivel de segunda clase. La sentencia final es el "else" definitivo, ya que TRUE siempre se evalúa como verdadero, por lo que a todos los registros no categorizados por la primera y la segunda línea se les asigna el nivel de tercera clase:

```
> titanic_train <- titanic_train |>
+   mutate(
+     fare_class = case_when(
+       Fare >= 31 ~ "1st Class",
+       Fare >= 15 ~ "2nd Class",
+       TRUE ~ "3rd Class"
```

```
+     )
+   )
```

La característica resultante tiene tres niveles, como se esperaba:

```
>  table(titanic_train$fare_class)

1st Class   2nd Class   3rd Class
    225         209         457
```

En caso de que no comprendamos el significado real de las tarifas, como las tarifas de primera, segunda y tercera clase, podríamos aplicar la heurística de discretización descrita anteriormente, que utiliza percentiles naturales o puntos de corte intuitivos de valores en lugar de grupos significativos.

La función cut() está incluida en R base y proporciona un método sencillo para crear un factor a partir de un vector numérico. El parámetro breaks especifica los puntos de corte para el rango numérico, que se muestra a continuación para un factor de tres niveles que coincide con la discretización anterior. El parámetro right = FALSE indica que los niveles no deben incluir el valor más a la derecha, o el más alto, y el punto de corte Inf indica que la categoría final puede abarcar el rango de valores de 31 a infinito. Las categorías resultantes son idénticas al resultado anterior, pero usan etiquetas diferentes:

```
> table(cut(titanic_train$Fare, breaks = c(0, 15, 31, Inf),
+   right = FALSE))

 [0,15)  [15,31) [31,Inf)
    457       209       225
```

Por defecto, cut() establece etiquetas para los factores que indican el rango de valores que corresponde a cada nivel.
Los corchetes indican que el número entre corchetes está incluido en el nivel, mientras que los paréntesis indican un número que no está incluido. Si se desea, se puede asignar un vector de etiquetas de factor al parámetro de etiquetas para el resultado.

La función cut() se vuelve más interesante al combinarse con una secuencia de valores generada por la función seq(). Aquí, creamos niveles para los 11 rangos de valores de 0 a 550 en incrementos de 50:

```
> table(cut(titanic_train$Fare, right = FALSE,
+   breaks = seq(from = 0, to = 550, by = 50)))
```

```
[0,50)  [50,100) [100,150) [150,200) [200,250) [250,300) [300,350) [350,400) [400,450) [450,500) [500,550)
   730       108        24         9        11         6         0         0         0         0         3
```

Usar intervalos de amplitud uniforme reduce la dimensionalidad, pero no resuelve el problema de la dispersión. Los dos primeros niveles contienen la mayoría de los ejemplos, pero el resto tiene muy pocos, o incluso cero en algunos casos.

Como alternativa a tener intervalos de igual tamaño, podemos construir contenedores (*bins*) con el mismo número de ejemplos. Hemos utilizado la función quantile() en temas anteriores para identificar los puntos de corte de quintiles y percentiles, pero aún sería necesario usar estos valores con una función cut() para crear los niveles factoriales. El siguiente código crea cinco intervalos para los quintiles, pero podría adaptarse para cuartiles, deciles o percentiles:

```
> table(cut(titanic_train$Fare, right = FALSE,
+   breaks = quantile(titanic_train$Fare,
+     probs = seq(0, 1, 0.20))))

[0,7.85) [7.85,10.5) [10.5,21.7) [21.7,39.7)  [39.7,512)
     166         173         196         174         179
```

Ten en cuenta que los contenedores no contienen exactamente el mismo número de ejemplos debido a la presencia de empates.
Tidyverse también incluye una función para crear grupos basados en cuantiles, que puede ser más fácil de usar en algunos casos. Esta función ntile() divide los datos en n grupos de igual tamaño. Por ejemplo, puede crear cinco grupos de la siguiente manera:

```
> table(ntile(titanic_train$Fare, n = 5))
  1   2   3   4   5
179 178 178 178 178
```

Dado que la función asigna etiquetas numéricas a los grupos, es importante convertir el vector resultante en un factor. Esto se puede hacer directamente con una sentencia mutate():

```
> titanic_train <- titanic_train |>
+   mutate(fare_level = factor(ntile(Fare, n = 11)))
```

La característica resultante tiene 11 niveles equitativamente proporcionados:

```
> table(titanic_train$fare_level)

 1  2  3  4  5  6  7  8  9 10 11
81 81 81 81 81 81 81 81 81 81 81
```

Aunque el nivel aún tiene etiquetas numéricas, dado que se ha codificado como un factor, la mayoría de las funciones de R lo tratarán como categórico. Por supuesto, sigue siendo importante encontrar el equilibrio adecuado entre demasiados y demasiados niveles.

Manejo de datos faltantes

Los conjuntos de datos didácticos utilizados para los ejemplos en temas anteriores rara vez presentaron el problema de datos faltantes, donde un valor que debería estar presente está ausente. El lenguaje R utiliza el valor especial NA para indicar estos valores faltantes, que la mayoría de las funciones de aprendizaje automático no pueden gestionar de forma nativa.

En el tema, Encontrar grupos de datos: Agrupamiento con k-medias, pudimos reemplazar los valores faltantes con una estimación del valor real basada en otra información disponible en el conjunto de datos mediante un proceso llamado imputación. Específicamente, los valores faltantes de edad de los estudiantes de secundaria se imputaron con la edad promedio de los estudiantes que se graduaron en el mismo año. Esto proporcionó una estimación razonable del valor de edad desconocido.

Los datos faltantes son un problema mucho mayor en los proyectos de aprendizaje automático del mundo real de lo que se esperaría dada su rareza hasta la fecha. Esto se debe no solo a que los proyectos del mundo real son más desordenados y complejos que los ejemplos de cursos universitarios. Además, a medida que los conjuntos de datos aumentan de tamaño (al incluir más filas o columnas), una proporción relativamente pequeña de datos faltantes causará más problemas, ya que es más probable que cualquier fila o columna contenga al menos un valor faltante. Por ejemplo, incluso si la tasa de valores faltantes es de tan solo el uno por ciento, en un conjunto de datos con 100 columnas, esperaríamos que la fila promedio tuviera un valor faltante. En este caso, simplemente excluir todas las filas con valores faltantes reduciría drásticamente el tamaño del conjunto de datos hasta el punto de eliminarlo por completo.

En campos como la economía, la bioestadística y las ciencias sociales, el enfoque de referencia para los datos faltantes es la **imputación múltiple**, que utiliza técnicas de modelado estadístico o aprendizaje automático para imputar todos los valores de las características faltantes a partir de los valores de las características no faltantes. Dado que esto tiende a disminuir la variabilidad de los datos y, por lo tanto, aumenta la certeza de las predicciones, el software moderno de imputación múltiple suele añadir variación aleatoria a los valores imputados para evitar sesgar las inferencias realizadas a partir del conjunto de datos. R cuenta con numerosos paquetes para realizar imputación múltiple, como:

- mice: Imputación multivariante mediante ecuaciones encadenadas.

- Amelia: Un programa para datos faltantes (nombrado en honor a la famosa piloto Amelia Earhart, quien desapareció en 1937 durante un intento por convertirse en la primera mujer piloto en dar la vuelta al mundo).
- Simputation: Imputación simple, que intenta simplificar el manejo de datos faltantes mediante el uso de funciones compatibles con tidyverse.
- missForest: Imputación no paramétrica de valores faltantes mediante Random Forest, un paquete que utiliza métodos de aprendizaje automático de vanguardia para imputar cualquier tipo de datos, incluso tipos con relaciones complejas y no lineales entre las características.

A pesar de la abundancia de herramientas de software de imputación múltiple, en comparación con los proyectos de estadística tradicional y ciencias sociales, los proyectos de aprendizaje automático aplican métodos más sencillos para gestionar datos faltantes. Esto se debe a que los objetivos y las consideraciones difieren.

Los proyectos de aprendizaje automático tienden a centrarse en métodos que funcionan con conjuntos de datos muy grandes y facilitan la predicción en un conjunto de prueba futuro e imprevisto, incluso si se violan ciertos supuestos estadísticos. Por otro lado, los métodos más formales de las ciencias sociales se centran en estrategias que tienden a ser más intensivas computacionalmente, pero que conducen a estimaciones imparciales para la inferencia y la prueba de hipótesis. Ten presente esta distinción al leer las secciones siguientes, que abarcan técnicas prácticas comunes para el manejo de datos faltantes, pero que generalmente no son recomendables para el análisis científico formal.

Comprendiendo los tipos de datos faltantes

No todos los datos faltantes se crean de la misma manera, y algunos tipos son más problemáticos que otros. Por esta razón, al preparar datos con valores faltantes, es útil considerar las razones subyacentes por las que falta un valor en particular. Imagina el proceso que generó el conjunto de datos y pregúntate por qué se dejaron ciertos valores en blanco. ¿Existe alguna razón lógica para que falten? ¿O se dejó en blanco por error o por pura casualidad? Responder a estas preguntas ayuda a encontrar la solución para reemplazar los valores faltantes de forma responsable. Las respuestas también distinguen tres tipos de datos faltantes, desde el menos problemático hasta el más grave:

1. Los datos **faltantes completamente aleatorios** (MCAR, *Missing completely at random*) son independientes de las demás características y de su propio valor; en otras palabras, no sería posible predecir si falta algún valor en particular. La falta puede deberse a un error aleatorio en la entrada de datos o a algún otro proceso que omite el valor aleatoriamente. Los datos faltantes completamente aleatorios pueden

concebirse como un proceso completamente impredecible que toma la matriz final de datos y selecciona aleatoriamente las celdas que se eliminarán.

2. Los datos **faltantes completamente aleatorios** (MAR, *Missing at random*) pueden depender de otras características, pero no del valor subyacente, lo que significa que ciertas filas predecibles tienen más probabilidades que otras de contener valores faltantes. Por ejemplo, los hogares en ciertas regiones geográficas pueden estar menos dispuestos a declarar sus ingresos familiares, pero suponiendo que lo hagan, lo hacen con honestidad. En esencia, MAR implica que los valores faltantes se seleccionan aleatoriamente tras controlar el factor o factores subyacentes que los causan.
3. Los datos **faltantes no aleatorios** (MNAR, *Missing not at random*) faltan debido a una razón relacionada con el propio valor faltante. Estos datos se eliminan del conjunto de datos por alguna razón imposible de discernir de las demás características. Por ejemplo, las personas con menos recursos pueden sentirse menos cómodas compartiendo sus ingresos, por lo que simplemente lo dejan en blanco. Otro ejemplo podría ser un sensor de temperatura que informa un valor faltante para temperaturas extremadamente altas o bajas. Es probable que la mayoría de los valores faltantes del mundo real sean MNAR, ya que suele haber algún mecanismo oculto y no medido que causa la falta. Muy pocas cosas son verdaderamente aleatorias en el mundo real.

Los métodos de imputación funcionan bien para los dos primeros tipos de datos faltantes. Aunque se podría pensar que los datos MCAR son los más difíciles de imputar debido a su independencia e imprevisibilidad, en realidad son el tipo ideal de datos faltantes para gestionar.

Aunque la información faltante es completamente aleatoria, los valores que se han ocultado aleatoriamente pueden ser predecibles dadas las demás características disponibles. Dicho de otro modo, la información faltante en sí es impredecible, pero los valores faltantes subyacentes pueden ser bastante predecibles. De igual manera, los datos MAR también son fácilmente predecibles debido a las características dadas.

Desafortunadamente, los datos NMAR, que son quizás el tipo más común de datos faltantes, son los menos susceptibles de predicción. Debido a que los valores faltantes fueron censurados por un proceso desconocido, cualquier modelo basado en estos datos tendrá una visión incompleta de la relación entre los datos faltantes y los no faltantes, y es probable que los resultados estén sesgados hacia los no faltantes.

Por ejemplo, supongamos que intentamos construir un modelo de impago de préstamos, y las personas más pobres son más propensas a dejar el campo de ingresos en blanco en la solicitud de préstamo. Si imputamos los ingresos faltantes, los valores imputados tenderán a ser mayores que los valores reales, ya que nuestra imputación se basó únicamente en los datos

disponibles, que omiten más valores bajos que altos. Si los hogares con ingresos más bajos tienen mayor probabilidad de impago, un modelo que utilice los valores de ingresos imputados sesgados para predecir los resultados de los préstamos subestimará la probabilidad de impago de los hogares que dejaron el campo de ingresos en blanco.

Debido a la posibilidad de dicho sesgo, en sentido estricto, solo deberíamos imputar datos MCAR y MAR.
Sin embargo, la imputación puede ser la menor de dos opciones imperfectas, ya que excluir filas con datos faltantes del conjunto de datos de entrenamiento también sesgará el modelo si los datos no faltan de forma completamente aleatoria. Por lo tanto, a pesar de violar los supuestos estadísticos, los profesionales del aprendizaje automático a menudo imputan valores faltantes en lugar de eliminar los datos faltantes del conjunto de datos. Las siguientes secciones muestran algunas estrategias comunes empleadas para este fin.

Imputación de valores faltantes

Dado que los valores NA no pueden ser gestionados directamente por muchas funciones de R ni por la mayoría de los algoritmos de aprendizaje automático, deben reemplazarse con otra cosa, idealmente, de una forma que mejore el rendimiento del modelo. En el aprendizaje automático, este tipo de imputación de valores faltantes supone una barrera para la predicción, lo que significa que se prefieren los enfoques más simples que funcionan razonablemente bien a los más complejos, incluso si estos últimos pueden ser más sólidos metodológica y teóricamente.

Los datos faltantes de tipo carácter pueden proporcionar la forma de datos faltantes con la solución más sencilla posible, ya que es posible tratar los valores faltantes como cualquier otro valor recodificando los valores NA en una cadena de caracteres literal como "Missing", "Unknown" u otra etiqueta de tu elección.

La cadena en sí es arbitraria; solo debe ser consistente para cada valor faltante dentro de la columna.
Por ejemplo, el conjunto de datos del Titanic incluye dos características categóricas con datos faltantes: Camarote y Embarcado. Podemos imputar fácilmente 'X' en lugar de los valores de Cabina faltantes y 'Unknown' en lugar de los valores de Embarcado faltantes de la siguiente manera:

```
> titanic_train <- titanic_train |>
+   mutate(
+     Cabin = if_else(is.na(Cabin), "X", Cabin),
+     Embarked = if_else(is.na(Embarked), "Unknown", Embarked)
+   )
```

Aunque este método ha eliminado los valores NA al reemplazarlos con cadenas de caracteres válidas, parece que deberían ser posibles enfoques más sofisticados. Después de todo, ¿no podríamos usar el aprendizaje automático para predecir valores faltantes utilizando las columnas restantes del conjunto de datos? De hecho, es posible, como veremos en breve. Sin embargo, usar este tipo de enfoque avanzado puede ser excesivo, o incluso perjudicial, para el rendimiento predictivo del modelo.

Las razones por las que se podría realizar la imputación de valores faltantes varían según la disciplina. En la estadística tradicional y las ciencias sociales, los modelos se utilizan a menudo para la inferencia y la comprobación de hipótesis, más que para la predicción y el pronóstico. Cuando se utilizan para la inferencia, es fundamental que las relaciones entre las características del conjunto de datos se preserven con el mayor cuidado posible, ya que los estadísticos buscan estimar y comprender con precisión la conexión individual de cada característica con el resultado de interés. Imputar valores arbitrarios en las ranuras faltantes puede distorsionar estas relaciones, especialmente si los valores faltantes no son completamente aleatorios.

En cambio, los enfoques más sofisticados utilizan la información disponible para imputar una estimación razonable del valor real, manteniendo el mayor número posible de filas de datos y garantizando al mismo tiempo la preservación de las importantes relaciones internas entre las características. En definitiva, esto aumenta la **potencia estadística** del análisis, lo que se relaciona con la capacidad de detectar patrones y probar hipótesis.

Por el contrario, los profesionales del aprendizaje automático suelen preocuparse menos por las relaciones internas entre las características de un conjunto de datos y centrarse más en la relación de estas con un resultado objetivo externo. Desde esta perspectiva, no hay una razón sólida para aplicar estrategias de imputación sofisticadas.

Dichos métodos no aportan información nueva que pueda utilizarse para predecir mejor el objetivo, ya que simplemente refuerzan los patrones internos. Por otro lado, al suponer que los datos no faltan aleatoriamente, puede ser menos útil centrarse en el valor específico que se imputará en una ranura faltante y, en cambio, centrar el esfuerzo en intentar utilizar la falta misma como predictor del objetivo.

Imputación simple con indicadores de valores faltantes

La práctica descrita en la sección anterior, en la que los valores categóricos faltantes se reemplazaban con una cadena arbitraria como "Missing" o "Unknown", es una forma de imputación simple, en la que los valores NA simplemente se reemplazan por un valor constante. Para las características numéricas, podemos utilizar un enfoque equivalente. Para cada característica con un valor faltante, se elige un valor para imputar en lugar de los valores

NA. Este puede ser una estadística de resumen como la media, la mediana o la moda, o un número arbitrario; el valor específico generalmente no importa.

Aunque el valor exacto generalmente no importa, el enfoque más común puede ser la **imputación de la media**, quizás debido a la práctica habitual de hacerlo en el campo de la estadística tradicional. Un enfoque alternativo es utilizar un valor del mismo orden de magnitud, pero fuera del rango de valores reales encontrados en los datos. Por ejemplo, para valores de edad faltantes en el rango de 0 a 100, puedes optar por imputar el valor -1 o 999.

Ten en cuenta que, independientemente del valor elegido, las estadísticas de resumen calculadas sobre la característica se verán distorsionadas por los valores imputados.

Además de imputar un valor en lugar del NA, es especialmente importante crear un **indicador de valor faltante** (MVI, *Missing value indicator*), que es una variable binaria que indica si se imputó el valor de la característica. Haremos esto para los valores de edad faltantes de los pasajeros del Titanic utilizando el siguiente código:

```
> titanic_train <- titanic_train |>
+   mutate(
+     Age_MVI = if_else(is.na(Age), 1, 0),
+     Age = if_else(is.na(Age), mean(Age, na.rm = TRUE), Age)
+   )
```

Tanto la característica imputada como el MVI deben incluirse como predictores en el modelo de aprendizaje automático. La ausencia de un valor suele ser un predictor importante del objetivo y, sorprendentemente, uno de los más sólidos. Esto puede no ser inesperado si se considera la simple creencia de que en el mundo real muy pocas cosas ocurren al azar; si falta un valor, probablemente exista una explicación.

Por ejemplo, en el conjunto de datos del Titanic, la ausencia de un valor podría indicar algo sobre el estatus social o los antecedentes familiares del pasajero.

De igual manera, alguien que se niega a declarar sus ingresos u ocupación en una solicitud de préstamo podría estar ocultando que gana muy poco, lo cual podría ser un predictor sólido de impago. Este hallazgo de que los valores faltantes son predictores interesantes también se aplica a grandes cantidades de datos faltantes; una mayor cantidad de datos faltantes puede generar predictores aún más interesantes.

Patrones de valores faltantes

Ampliando la creencia de que un valor faltante puede ser un predictor muy útil, cada valor faltante adicional puede contribuir a nuestra capacidad para pronosticar un resultado específico.

En el caso de los datos de una solicitud de préstamo, una persona con datos faltantes sobre una sola característica podría haber ocultado su respuesta intencionalmente o simplemente haber omitido accidentalmente esta pregunta en el formulario de solicitud. Si una persona tiene datos faltantes sobre varias características, esta última excusa ya no aplica, o quizás implica que se apresuró en la solicitud o que, en general, es más irresponsable.

Podríamos asumir que las personas con más datos faltantes tienen mayor probabilidad de incumplir, pero, por supuesto, no lo sabremos hasta que entrenemos el modelo; también podría darse el caso de que las personas con más datos faltantes tengan, de hecho, menor probabilidad de incumplir, quizás porque algunas de las preguntas de la solicitud de préstamo no se aplican a su situación.

Supongamos que existe un patrón entre los registros con una gran cantidad de valores faltantes, pero que no se basa únicamente en el número de valores faltantes, sino en las características específicas que faltan.

Por ejemplo, alguien que teme declarar sus ingresos por ser demasiado bajos puede omitir preguntas relacionadas en una solicitud de préstamo, mientras que el propietario de una pequeña empresa puede omitir una sección diferente de preguntas sobre historial laboral porque no corresponden a un trabajador autónomo. Ambos casos pueden tener un número aproximadamente igual de valores faltantes, pero sus patrones pueden diferir sustancialmente.

Se puede construir un **patrón de valores faltantes** (MVP, *Missing value pattern*) para capturar estos comportamientos y utilizarlos como características para el aprendizaje automático. Un patrón de valores faltantes es esencialmente una cadena de caracteres compuesta por una serie de MVI, donde cada carácter de la cadena representa una característica con valores faltantes.

La figura 12 ilustra cómo funciona el proceso para un conjunto de datos simplificado de una solicitud de préstamo. Para cada una de las ocho características, construimos un indicador de valor faltante que indica si faltaba la celda correspondiente:

married	age	gender	country	emp_len	income	zipcode	creditscore
NA	*NA*	*NA*	USA	10	$54,000	90210	670
Yes	48	M	USA	6	$99,000	46656	780
Yes	37	F	USA	5	*NA*	*NA*	*NA*
No	*NA*	M	*NA*	*NA*	*NA*	32817	*NA*

x1_mvi	x2_mvi	x3_mvi	x4_mvi	x5_mvi	x6_mvi	x7_mvi	x8_mvi
1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1
0	1	0	1	1	1	0	1

Figura 12: La construcción de un patrón de valores faltantes comienza con la creación de indicadores de valor faltante para cada característica con datos faltantes.

Estos MVI binarios se concatenan en una sola cadena. Por ejemplo, la primera fila estaría representada por la cadena '11100000', que indica que faltaban las tres primeras características de este solicitante de préstamo. El segundo solicitante, sin datos faltantes, estaría representado por '00000000', mientras que el segundo y el tercero estarían representados por '00000111' y '01011101', respectivamente. El vector de caracteres R mvp resultante se convertiría en un factor para que un algoritmo de aprendizaje lo utilizara para la predicción. Cada nivel del factor representa un patrón específico de falta de datos; los solicitantes de préstamo que siguen el mismo patrón podrían tener resultados similares.

Aunque los patrones de valores faltantes pueden ser predictores extremadamente potentes, no están exentos de desafíos. En primer lugar, en un conjunto de datos con k características, existen 2^k valores potenciales de un patrón de valores faltantes. Un conjunto de datos con tan solo 10 características puede tener hasta 1,024 niveles del predictor MVP, mientras que uno con 25 características tendría más de 33 millones de niveles potenciales. Un conjunto de datos relativamente pequeño con 50 características tendría un número casi incontable de niveles potenciales de MVP, lo que haría al predictor inútil para el modelado.

A pesar de este posible problema, la esperanza con el enfoque MVP es que el número potencialmente elevado de niveles evite la maldición de la dimensionalidad debido a patrones de valores faltantes que distan mucho de ser uniformes o aleatorios. En otras palabras, el enfoque MVP depende en gran medida de que los datos no presenten valores faltantes de forma aleatoria; esperamos que exista un patrón subyacente sólido que impulse los valores faltantes, el cual se reflejará en los patrones de valores faltantes.

En general, cuanto menor sea la heterogeneidad presente en los valores faltantes, mayor será la frecuencia con la que ciertos patrones de valores faltantes aparezcan en los datos.

Desafortunadamente, incluso si una característica falta completamente al azar, puede reducir la utilidad del enfoque MVP, ya que, aunque las filas sean similares en casi todos los indicadores binarios de valores faltantes, si una sola difiere, se tratará como un patrón de valores faltantes completamente diferente. Para abordar este problema, una alternativa es utilizar el conjunto de datos MVI con un algoritmo de agrupamiento no supervisado, como el algoritmo de k-medias, para crear grupos de personas con patrones similares de valores faltantes.

El problema de los datos desequilibrados

Uno de los problemas de datos más desafiantes es el **desequilibrio**, que se produce cuando uno o más niveles de clase son mucho más comunes que los demás. Muchos algoritmos de aprendizaje automático, si no la mayoría, tienen grandes dificultades para aprender conjuntos de datos muy desequilibrados, y aunque no existe un umbral específico que determine cuándo un conjunto de datos está demasiado desequilibrado, los problemas causados por la falta de equilibrio se agravan a medida que el problema se agrava.

En las primeras etapas del desequilibrio de clases, se detectan pequeños problemas. Por ejemplo, medidas de rendimiento simples como la precisión comienzan a perder relevancia y se necesitan medidas de rendimiento más sofisticadas, como las descritas en el documento, Evaluación del rendimiento del modelo. A medida que el desequilibrio se amplía, surgen problemas mayores.

Por ejemplo, con conjuntos de datos extremadamente desequilibrados, algunos algoritmos de aprendizaje automático podrían tener dificultades para predecir el grupo minoritario. Teniendo esto en cuenta, sería prudente comenzar a preocuparse por los datos desequilibrados cuando la división es peor que 80% frente a 20%, preocuparse más cuando es peor que 90% frente a 10%, y asumir lo peor cuando la división es más severa que 99% frente a 1%.

Un desequilibrio de clases también puede ocurrir si el costo de clasificación errónea en el mundo real de uno o más niveles de clase es significativamente mayor o menor que el de los demás.

Los datos desequilibrados son un desafío frecuente pero importante porque muchos de los resultados del mundo real que queremos predecir son significativos, principalmente porque son poco frecuentes y costosos. Esto incluye tareas de predicción para identificar resultados como:

- Enfermedades graves
- Clima extremo y desastres naturales

- Actividad fraudulenta
- Incumplimiento de préstamos
- Fallas de hardware o mecánicas
- Clientes con alto poder adquisitivo o los llamados “clientes ballena”

Como pronto aprenderás, lamentablemente no existe una única forma óptima de abordar problemas de clasificación desequilibrada como estos, e incluso las técnicas más avanzadas tienen sus inconvenientes.
Quizás el enfoque más importante sea ser consciente del problema de los datos desequilibrados, reconociendo que todas las soluciones son imperfectas.

Estrategias sencillas para reequilibrar datos

Si un conjunto de datos presenta un desequilibrio grave, con algunos niveles de clase con demasiados o muy pocos ejemplos, una solución sencilla consiste en restar ejemplos de las clases mayoritarias o añadir ejemplos de las clases minoritarias. La primera estrategia se denomina **submuestreo** (*undersampling*) y, en su caso más simple, consiste en descartar registros aleatoriamente de las clases mayoritarias. El segundo enfoque se denomina **sobremuestreo** (*oversampling*). Idealmente, se recopilarían más filas de datos, pero esto no suele ser posible. En su lugar, se duplican aleatoriamente los ejemplos de las clases minoritarias hasta alcanzar el equilibrio de clases deseado.

Tanto el submuestreo como el sobremuestreo presentan importantes inconvenientes, pero pueden ser eficaces en determinadas circunstancias. El principal peligro del submuestreo es el riesgo de descartar ejemplos que expresan patrones pequeños pero importantes en los datos. Por lo tanto, el submuestreo funciona mejor si el conjunto de datos es lo suficientemente grande como para reducir el riesgo de que la eliminación de una parte sustancial de las clases mayoritarias excluya por completo los ejemplos de entrenamiento clave. Además, en la era del big data, siempre genera una sensación de derrota entregar información voluntariamente.

El sobremuestreo evita esta decepción al generar ejemplos adicionales de clases minoritarias, pero corre el riesgo de sobreajustar a patrones poco importantes o ruido en los casos minoritarios. Tanto el submuestreo como el sobremuestreo se han incluido en enfoques de **muestreo focalizado** más avanzados que evitan el muestreo aleatorio simple y priorizan los registros que maximizan los límites de decisión entre los grupos.

Estas técnicas rara vez se utilizan en la práctica debido a su ineficiencia computacional y su limitada eficacia en el mundo real.

Para una revisión más detallada de las estrategias para el manejo de datos desequilibrados, consulta: Data Mining for Imbalanced Datasets: An Overview., Chawla, N., 2010, en Data Mining and Knowledge Discovery Handbook, 2nd Edition, Maimon, O. y Rokach, L.

Para ilustrar las técnicas de remuestreo, volveremos al conjunto de datos de redes sociales de adolescentes utilizado anteriormente en este documento (primera parte) y comenzaremos cargándolo y preparándolo con varios comandos de tidyverse.

Primero, utilizando las funciones fct_ del paquete forcats, la característica de género se recodifica como un factor con las etiquetas Male y Female, y los valores NA se recodifican como Unknown. A continuación, las edades (ages) atípicas menores de 13 años o mayores de 20 años se reemplazan por valores NA. A continuación, utilizando group_by() en combinación con mutate(), podemos imputar las edades faltantes con la mediana de edad por año de graduación.

Por último, desagrupamos los datos y reordenamos las columnas con select() de forma que las características de interés aparezcan primero en el conjunto de datos. El comando completo es el siguiente:

```
> library(tidyverse)
> library(forcats)
> snsdata <- read_csv("snsdata.csv") |>
+   mutate(
+     gender = fct_recode(gender, Female = "F", Male = "M"),
+     gender = fct_na_value_to_level(gender, level = "Unknown"),
+     age = ifelse(age < 13 | age > 20, NA, age)
+   ) |>
+  group_by(gradyear) |>
+  mutate(age_imp = if_else(is.na(age),
+     median(age, na.rm = TRUE), age)) |>
+  ungroup() |>
+  select(gender, friends, gradyear, age_imp, basketball:drugs)
```

En este conjunto de datos, los hombres y las personas de género desconocido están subrepresentados, lo cual podemos confirmar con la función fct_count():

```
> fct_count(snsdata$gender, prop = TRUE)

# A tibble: 3 × 3
  f           n      p
  <fct>   <int>  <dbl>
1 Female  22054 0.735
2 Male     5222 0.174
3 Unknown  2724 0.0908
```

Una estrategia sería submuestrear los grupos de mujeres y hombres de forma que los tres tengan el mismo número de registros. El paquete caret, presentado por primera vez en el documento, Evaluación del Rendimiento del Modelo, incluye la función downSample() que permite realizar esta técnica. El parámetro *y* es la característica categórica con los niveles que se equilibrarán, el parámetro *x* especifica las columnas restantes que se incluirán en el frame de datos remuestreado, y el parámetro yname es el nombre de la columna de destino:

```
> library(caret)
> sns_undersample <- downSample(x = snsdata[2:40],
+   y = snsdata$gender,
+   yname = "gender")
```

El conjunto de datos resultante incluye 2,724 ejemplos de cada uno de los tres niveles de clase:

```
> fct_count(sns_undersample$gender, prop = TRUE)

# A tibble: 3 x 3
  f           n     p
  <fct>   <int> <dbl>
1 Female   2724 0.333
2 Male     2724 0.333
3 Unknown  2724 0.333
```

La función upSample() del paquete caret realiza un sobremuestreo, de modo que los tres niveles tengan el mismo número de ejemplos que la clase mayoritaria:

```
> sns_oversample <- upSample(x = snsdata[2:40],
+   y = snsdata$gender,
+   yname = "gender")
```

El conjunto de datos resultante incluye 22,054 ejemplos de cada una de las tres categorías de género:

```
> fct_count(sns_oversample$gender, prop = TRUE)

# A tibble: 3 x 3
  f           n     p
  <fct>   <int> <dbl>
1 Female  22054 0.333
2 Male    22054 0.333
3 Unknown 22054 0.333
```

La mejor eficacia del enfoque de sobremuestreo o submuestreo depende del conjunto de datos y del algoritmo de aprendizaje automático utilizado. Sería conveniente construir modelos

entrenados con conjuntos de datos creados mediante cada una de estas técnicas de remuestreo y observar cuál ofrece un mejor rendimiento en las pruebas.

Sin embargo, es fundamental tener en cuenta que las medidas de rendimiento deben calcularse en un conjunto de pruebas desequilibrado; la evaluación debe reflejar el desequilibrio de la clase original, ya que así es como el modelo deberá funcionar en condiciones reales.

Generación de un conjunto de datos sintético equilibrado con SMOTE

Además del submuestreo y el sobremuestreo, un tercer enfoque de reequilibrio, denominado **generación sintética**, crea nuevos ejemplos de la clase minoritaria con el objetivo de reducir la tendencia del sobremuestreo a sobreajustar los ejemplos de la clase minoritaria. Actualmente, existen numerosos métodos de reequilibrio de generación sintética, pero uno de los primeros en adquirir gran popularidad fue el algoritmo **SMOTE**, introducido por Chawla et al.

En 2002, su nombre hace referencia al uso de una técnica sintética de sobremuestreo de minorías. En resumen, el algoritmo utiliza un conjunto de heurísticas para construir nuevos registros similares, pero no exactamente iguales, a los observados previamente. Para construir registros similares, SMOTE utiliza el concepto de similitud descrito en el documento, Aprendizaje perezoso: clasificación mediante vecinos más cercano, y, de hecho, utiliza directamente aspectos del enfoque k-NN.

Para más información sobre el algoritmo SMOTE, consultar: SMOTE: Synthetic Minority Over-Sampling Technique, 2002, Chawla, N., Bowyer, K., Hall, L. y Kegelmeyer, W., Journal of Artificial Intelligence Research, Vol. 16, pp. 321-357.

Para comprender el funcionamiento de SMOTE, supongamos que queremos sobremuestrear una clase minoritaria de modo que el conjunto de datos resultante contenga el doble de ejemplos de esta clase. En el caso del sobremuestreo estándar, simplemente duplicaríamos cada registro minoritario para que aparezca dos veces. En técnicas de sobremuestreo sintético como SMOTE, en lugar de duplicar cada registro, crearíamos un nuevo registro sintético. Si se desea un sobremuestreo mayor o menor, simplemente generamos más o menos de un nuevo registro sintético por cada registro original.

Queda una pregunta: ¿cómo se construyen exactamente los registros sintéticos? Aquí es donde entra en juego la técnica de k-vecinos más próximos. El algoritmo encuentra los k vecinos más cercanos de cada una de las observaciones originales de la clase minoritaria. Por convención, k suele establecerse en cinco, pero se puede establecer un valor mayor o menor

si se desea. Para cada registro sintético que se crea, el algoritmo selecciona aleatoriamente uno de los k vecinos más cercanos de las observaciones originales.

Por ejemplo, para duplicar la clase minoritaria, seleccionaría aleatoriamente un vecino más cercano de cinco para cada una de las observaciones originales; para triplicar los datos originales, se seleccionarían dos de los cinco vecinos más cercanos para cada observación, y así sucesivamente.

Dado que la selección aleatoria de los vecinos más cercanos simplemente copia los datos originales, se necesita un paso más para generar observaciones sintéticas. En este paso, el algoritmo identifica el vector entre cada una de las observaciones originales y sus vecinos más cercanos elegidos aleatoriamente. Se elige un número aleatorio entre 0 y 1 para reflejar la proporción de distancia a lo largo de esta línea para colocar el punto de datos sintéticos.

Los valores de las características de este punto serán entre el 100 % idénticos a los valores de las características de la observación original y el 100 % idénticos a los valores de las características del vecino, o cualquier valor intermedio. Esto se representa en la siguiente figura, que ilustra cómo las observaciones sintéticas pueden colocarse aleatoriamente en las líneas que conectan los cuatro puntos de datos originales con sus vecinos. Añadir las seis observaciones sintéticas crea un equilibrio mucho mejor de las clases de círculo y cuadrado y, por lo tanto, fortalece el límite de decisión y, potencialmente, facilita que un algoritmo de aprendizaje descubra el patrón.

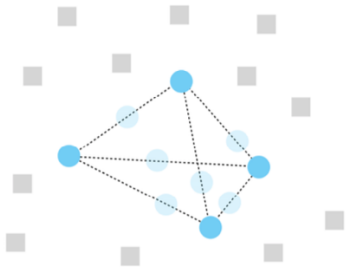

Figura 13: SMOTE puede crear seis observaciones sintéticas a partir de cuatro observaciones minoritarias originales, lo que refuerza la frontera de decisión entre las dos clases (círculos y cuadrados).

Por supuesto, la dependencia del algoritmo SMOTE de los vecinos más cercanos y el uso de funciones de distancia implica que se aplican las mismas advertencias de preparación de datos que con k-NN. En primer lugar, el conjunto de datos debe ser completamente numérico.

En segundo lugar, aunque no es estrictamente necesario, puede ser conveniente transformar los valores numéricos de las características para que se encuentren en la misma escala, de modo que los rangos amplios no dominen la selección de vecinos más cercanos. Veremos esto en la práctica en la siguiente sección.

Ejemplo - Aplicación del algoritmo SMOTE en R

Existen varios paquetes de R que incluyen implementaciones del algoritmo SMOTE. El paquete DMwR incluye la función SMOTE(), tema de numerosos tutoriales, pero actualmente no está disponible para las versiones recientes de R.

El paquete smotefamily incluye diversas funciones SMOTE y está bien documentado, pero no se ha actualizado en varios años. Por lo tanto, usaremos la función smote() del paquete themis (https://themis.tidymodels.org), que recibe su nombre de Temis, la diosa griega de la justicia, a quien se suele representar sosteniendo una balanza. Este paquete es fácil de usar y está bien integrado en tidyverse. Para ilustrar la sintaxis básica de la función smote(), comenzaremos canalizando el conjunto de datos snsdata y utilizando el género como la característica a equilibrar:

```
> library(themis)
> sns_balanced <- snsdata |> smote("gender")
```

Para verificar el resultado, utilizamos la función table() en el conjunto de datos, que aumentó de 30,000 filas a 66,162, pero ahora está equilibrado en las tres categorías de género:

```
> table(sns_balanced$gender)
 Female    Male.  Unknown
  22054   22054    22054
```

Aunque esto generó un equilibrio de género, dado que el algoritmo SMOTE se basa en los vecinos más cercanos, determinados mediante cálculos de distancia, puede ser mejor normalizar los datos antes de generar los datos sintéticos. Por ejemplo, dado que la característica de amigos varía de 0 a 830, mientras que la de fútbol solo varía de 0 a 15, es probable que los vecinos más cercanos se inclinen por aquellos con un número de amigos similar en lugar de intereses similares. Aplicar la normalización de mínimos y máximos puede ayudar a mitigar estos problemas al reescalar todas las características para que tengan un rango entre 0 y 1.

Ya hemos escrito nuestra propia función de normalización, que implementaremos de nuevo aquí:

```
> normalize <- function(x) {
+   return ((x - min(x)) / (max(x) - min(x)))
+ }
```

Para restaurar los datos a su escala original, también necesitaremos la función unnormalize(). Como se define aquí, la función toma dos parámetros: el primero es un vector, norm_values, que almacena los valores normalizados; el segundo es una cadena con el nombre de la columna normalizada. Necesitamos este nombre de columna para obtener los valores mínimo y máximo de esta columna a partir de los datos originales sin normalizar del conjunto de datos snsdata. El vector unnormalized_vals resultante utiliza estos valores mínimo y máximo para revertir la normalización, y luego los valores se redondean a enteros como en los datos originales, excepto la característica age_imp, que originalmente era un decimal.

La función unnormalize() completa es la siguiente:

```
> unnormalize <- function(norm_vals, col_name) {
+   old_vals <- snsdata[col_name]
+   unnormalized_vals <- norm_vals *
+     (max(old_vals) - min(old_vals)) + min(old_vals)
+   rounded_vals <- if(col_name != "age_imp")
+     { round(unnormalized_vals) }
+     else {unnormalized_vals}
+   return (rounded_vals)
+ }
```

Con una secuencia de barras verticales, podemos aplicar la normalización antes de usar la función smote() y, posteriormente, la desnormalización. Esto utiliza la función dplyr across() para normalizar y desnormalizar las columnas con datos numéricos. En el caso de la función unnormalize(), la sintaxis es ligeramente más compleja debido al uso de una lambda, representada por la tilde (~), que define una función que se utilizará en las columnas con datos numéricos.

La función normalize() no requería el uso de una lambda porque solo usa un parámetro, mientras que unnormalize() usa dos. El .x hace referencia al vector de datos de la columna y se pasa como primer parámetro, mientras que la función cur_column() se usa para pasar el nombre de la columna actual como segundo parámetro. La secuencia completa de comandos es la siguiente:

```
> snsdata_balanced <- snsdata |>
+   mutate(across(where(is.numeric), normalize)) |>
+   smote("gender") |>
+   mutate(across(where(is.numeric), ~unnormalize(.x, cur_column())))
```

Como antes, al comparar el equilibrio de género antes y después de SMOTE, observamos que las categorías ahora son equivalentes:

```
> table(snsdata$gender)
 Female    Male Unknown
  22054    5222    2724

> table(snsdata_balanced$gender)
 Female    Male Unknown
  22054   22054   22054
```

Observa que ahora tenemos más de cuatro veces más hombres y más de ocho veces más registros con género desconocido, o que se han añadido aproximadamente tres registros sintéticos masculinos y siete registros sintéticos con género desconocido por cada registro original con género masculino o desconocido, respectivamente. El número de ejemplos femeninos se ha mantenido igual.

Este conjunto de datos equilibrado ahora puede utilizarse con algoritmos de aprendizaje automático, teniendo en cuenta que el modelo se basará principalmente en casos sintéticos en lugar de ejemplos “reales” de las clases minoritarias. Que esto mejore el rendimiento puede variar de un proyecto a otro, por razones que se explican en la siguiente sección.

Considerando si un conjunto de datos equilibrado siempre es mejor

Si bien es innegable que los conjuntos de datos gravemente desequilibrados plantean desafíos para los algoritmos de aprendizaje, el mejor enfoque para gestionar el desequilibrio no está del todo claro. Algunos incluso argumentan que el mejor enfoque es no hacer nada. La cuestión es si equilibrar artificialmente el conjunto de datos puede mejorar el rendimiento general de un algoritmo de aprendizaje o si simplemente se trata de intercambiar una reducción en la especificidad por una mejora en la sensibilidad. Dado que un algoritmo de aprendizaje entrenado con un conjunto de datos equilibrado artificialmente se implementará algún día en el conjunto de datos original, desequilibrado, parece que la práctica de equilibrar consiste simplemente en ajustar la percepción del aprendiz sobre el costo de un tipo de error frente al otro.

Por lo tanto, resulta contradictorio comprender cómo descartar datos podría resultar en un modelo más inteligente, es decir, uno con mayor capacidad para distinguir realmente entre los resultados.

Entre quienes se muestran escépticos ante el equilibrio artificial de los datos de entrenamiento, el prolífico bioestadístico Frank Harrell ha escrito extensamente sobre este tema. En una reflexiva entrada de blog, escribió:

> "Los usuarios de clasificadores automáticos saben que una muestra muy desequilibrada con respecto a una variable de resultado binaria Y resulta en un clasificador extraño… Por esta razón, se utiliza la práctica inusual de submuestrear los controles para intentar equilibrar las frecuencias y obtener alguna variación que conduzca a clasificadores con un aspecto sensato (los usuarios de modelos de regresión nunca excluirían datos válidos para obtener una respuesta). Luego, tienen que, de alguna manera imprecisa, construir el clasificador para compensar el sesgo de la muestra".

Claramente, Harrell no cree que equilibrar la muestra sea, en general, un enfoque inteligente.

Para más artículos de Harrell sobre este tema, consulta:
http://www.fharrell.com/post/classification/ y
http://www.fharrell.com/post/class-damage/.

Nina Zumel, autora de libros sobre ciencia de datos, realizó experimentos para determinar si equilibrar artificialmente el conjunto de datos mejoraba el rendimiento de la clasificación. Tras realizar un experimento, concluyó que:

> "La clasificación suele ser más sencilla cuando las clases están casi equilibradas… Sin embargo, siempre he sido escéptica ante la afirmación de que equilibrar artificialmente las clases siempre sea útil cuando el modelo se ejecuta en una población con las prevalencias de clases nativas… El equilibrio de las clases, o el enriquecimiento en general, tiene un valor limitado si el objetivo es aplicar etiquetas de clase… no es una buena idea para los modelos de regresión logística".

Al igual que Frank Harrell, Nina Zumel también duda de la necesidad de equilibrar artificialmente los conjuntos de datos para los modelos de clasificación. Sin embargo, ambas perspectivas se oponen a la evidencia empírica y anecdótica que sugiere que equilibrar artificialmente un conjunto de datos, de hecho, mejora el rendimiento de un modelo.

Para una descripción completa del experimento de Zumel sobre la clasificación de datos desequilibrados, consultar:
https://win vector.com/2015/02/27/does-balancing-classes-improveclassifier-performance/.

¿Qué explica este resultado contradictorio? Podría estar relacionado con la elección de la herramienta. Los algoritmos de aprendizaje estadístico, como la regresión, pueden estar bien calibrados, lo que significa que estiman eficazmente las verdaderas probabilidades subyacentes de un resultado, incluso para resultados poco frecuentes.

Muchos algoritmos de aprendizaje automático, como los árboles de decisión y el método naïve Bayes, no están bien calibrados y, por lo tanto, podrían necesitar ayuda mediante un balanceo artificial para generar probabilidades razonables.

Independientemente de si se emplea una estrategia de balanceo, es importante utilizar un enfoque de evaluación del modelo que refleje el desequilibrio natural con el que se espera que el modelo se comporte durante la implementación. Esto implica priorizar medidas que consideren los costos, como kappa, sensibilidad y especificidad, o precisión y recuperación, así como un examen de la curva ROC (curva característica operativa del receptor), como se explica en el documento, Evaluación del rendimiento del modelo.

Si bien es recomendable ser escéptico ante el balanceo artificial del conjunto de datos, también puede ser útil intentarlo en los problemas de datos más complejos.

Resumen

Este documento pretendía exponerte a varios tipos nuevos de datos desafiantes que, aunque no se encuentran con frecuencia en ejemplos didácticos sencillos, se encuentran con frecuencia en la práctica. A pesar de los refranes populares que dicen que “nunca se tiene demasiado de algo bueno” o que “más siempre es mejor”, esto no siempre ocurre con los algoritmos de aprendizaje automático, que pueden distraerse con datos irrelevantes o tener dificultades para encontrar la aguja en el pajar si se ven abrumados por detalles menos importantes.

Una de las aparentes paradojas de la llamada era del big data es que una mayor cantidad de datos es, al mismo tiempo, lo que hace posible el aprendizaje automático y lo que lo dificulta; de hecho, un exceso de datos puede incluso conducir a la llamada “maldición de la dimensionalidad”.

Por muy decepcionante que sea desperdiciar parte del tesoro del big data, a veces es necesario para que el algoritmo de aprendizaje funcione como se desea. Quizás sea mejor considerar esto como una curación de datos, en la que se resaltan los detalles más relevantes. Las técnicas de reducción de dimensionalidad, como la selección y extracción de características, son importantes para algoritmos que no cuentan con métodos de selección integrados, pero también ofrecen beneficios como una mayor eficiencia computacional, que puede ser un cuello de botella clave para grandes conjuntos de datos. Los datos dispersos también requieren ayuda para que el algoritmo de aprendizaje detecte los detalles importantes, al igual que ocurre con los problemas de valores atípicos y datos faltantes.

Los datos faltantes, que hasta ahora solo han sido un problema menor en el curso, representan un desafío significativo en muchos conjuntos de datos del mundo real. Los profesionales del aprendizaje automático suelen elegir el enfoque más simple para resolver el problema (es decir, el que requiere menos trabajo para que el modelo funcione razonablemente bien); sin embargo, se están utilizando enfoques basados en el aprendizaje automático, como la

imputación múltiple, para crear conjuntos de datos completos en los campos de la estadística tradicional, la bioestadística y las ciencias sociales.

El problema de los datos desequilibrados puede ser el tipo de datos desafiantes más difícil de abordar.

Muchas de las aplicaciones más importantes del aprendizaje automático implican predicciones sobre conjuntos de datos desequilibrados, pero no existen soluciones fáciles, solo compromisos. Técnicas como el sobremuestreo y el submuestreo son sencillas, pero presentan importantes desventajas. Mientras que técnicas más complejas como SMOTE son prometedoras, pero pueden presentar nuevos problemas, y la comunidad está dividida sobre cuál es el mejor enfoque.

La lección más importante, en cualquier caso, es asegurar que la estrategia de evaluación refleje las condiciones que el modelo encontrará durante la implementación.

Por ejemplo, incluso si el modelo se entrena con un conjunto de datos equilibrado artificialmente, debe probarse y evaluarse utilizando el equilibrio natural de resultados.
Tras estos desafíos con los datos, lo que sigue es la construcción de modelos.

Si bien la preparación de datos es un componente fundamental para formar mejores estudiantes (después de todo, la información que entra genera información que sale), hay mucho más que podemos hacer para mejorar el proceso de aprendizaje. Sin embargo, estas técnicas requerirán más que un algoritmo estándar; requerirán creatividad y determinación para maximizar el potencial de los estudiantes.