Resumen del capítulo: clasificación desequilibrada

Ajuste de peso de clase

Los algoritmos de aprendizaje automático consideran que todas las observaciones del conjunto de entrenamiento tienen la misma ponderación de forma predeterminada. Si necesitamos indicar que algunas observaciones son más importantes, asignamos un peso a la clase respectiva.

La regresión logística, el árbol de decisión y el bosque aleatorio en la librería sklearn tienen un argumento class_weight. De forma predeterminada, es `None`, es decir, las clases son equivalentes:

`clase "0" peso = 1.0`

`clase "1" peso = 1.0`

Si especificamos `class_weight='balanced'`, el algoritmo calculará cuántas veces la clase "0" ocurre con más frecuencia que la clase "1". Denotaremos este número como N (un número desconocido de veces). Los nuevos pesos de clase se ven así:

`clase "0" peso = 1.0`

`clase "1" peso = N`

La clase rara tendrá un mayor peso.

Sobremuestreo

Cuando entrenamos modelos, las clases se pueden equilibrar aumentando el tamaño de la muestra. La técnica se llama **sobremuestreo.**

El sobremuestreo se realiza en varios pasos:

- Divide la muestra de entrenamiento por clase.
- Determina la clase con menos observaciones. Llámala la clase rara.
- Duplica las observaciones de clase más raras varias veces.
- Crea una nueva muestra de entrenamiento basada en los datos obtenidos.
- Mezcla los datos

Se pueden copiar observaciones varias veces usando la sintaxis de multiplicación de listas de Python. Para repetir los elementos de la lista, la lista se multiplica por un número (el número requerido de repeticiones):

```
answers = [0, 1, 0]
print(answers)
answers_x3 = answers * 3
print(answers_x3)
```

```
[0, 1, 0]
[0, 1, 0, 0, 1, 0, 0, 1, 0]
```

Utiliza la función **pd.concat()** para concatenar las tablas. Esta toma una lista de tablas como entrada. Los datos de origen se pueden obtener de la siguiente manera:

```
pd.concat([table1, table2])
```

Para mezclar las observaciones de forma aleatoria, usa el método *shuffle* de la librería *sklearn.utils:*

```
features, target = shuffle(features, target, random_state=54321)
```

La función upsample():

```
def upsample(features, target, repeat):
    features_zeros = features[target == 0]
    features_ones = features[target == 1]
    target_zeros = target[target == 0]
    target_ones = target[target == 1]

    features_upsampled = pd.concat([features_zeros] + [features_ones] * repeat)
    target_upsampled = pd.concat([target_zeros] + [target_ones] * repeat)

    features_upsampled, target_upsampled = shuffle(
        features_upsampled, target_upsampled, random_state=54321)

    return features_upsampled, target_upsampled
```

Submuestreo

En lugar de repetir las preguntas importantes, también podemos eliminar una parte de las que no son importantes. Para ello, podemos utilizar la técnica de **submuestreo.**

El submuestreo se realiza en varios pasos:

- Divide la muestra de entrenamiento por clase
- Determina la clase con más observaciones. Llámala la clase mayoritaria
- Elimina de forma aleatoria una parte de las observaciones de la clase mayoritaria
- Crea una nueva muestra de entrenamiento basada en los datos obtenidos
- Mezcla los datos

Para descartar de forma aleatoria algunos de los elementos de la tabla, usa la función *sample().* Toma *frac (de fracción)* y devuelve elementos aleatorios en cantidades tales que su fracción en la tabla inicial sea igual a *frac.*

```
features_sample = features_train.sample(frac=0.1, random_state=54321)
```

La función *downsample()*:

```
def downsample(features, target, fraction):
    features_zeros = features[target == 0]
    features_ones = features[target == 1]
    target_zeros = target[target == 0]
    target_ones = target[target == 1]

    features_downsampled = pd.concat(
        [features_zeros.sample(frac=fraction, random_state=54321)] + [features_ones])
    target_downsampled = pd.concat(
        [target_zeros.sample(frac=fraction, random_state=54321)] + [target_ones])

    features_downsampled, target_downsampled = shuffle(
        features_downsampled, target_downsampled, random_state=54321)

    return features_downsampled, target_downsampled
```

Umbral de clasificación

Para determinar la respuesta, la regresión logística calcula la *probabilidad de las clases.* Solo tenemos dos clases (cero y uno). La probabilidad de la clase "1" será suficiente para nosotros. El valor está en el rango de cero a uno: si es mayor a 0.5, la observación es positiva; de lo contrario, es negativa.

La línea donde termina la clase negativa y comienza la clase positiva se llama umbral. Por defecto, es 0.5, pero podemos cambiarlo.

Ajuste de umbral

En *sklearn*, la probabilidad de clase se puede calcular con la función **predict_proba()**. Lo que hace es tomar características de las observaciones y devolver las probabilidades:

```
probabilities = model.predict_proba(features)
```

Las cadenas corresponden a las observaciones. La primera columna indica la probabilidad de clase negativa y la segunda indica la probabilidad de clase positiva (las dos probabilidades suman la unidad).

Para crear un bucle con el rango deseado, usamos la función *arange()* de la librería numpy. Al igual que la función *range(),* esta itera sobre los elementos especificados del rango, pero es diferente porque funciona con números fraccionarios, además de enteros:

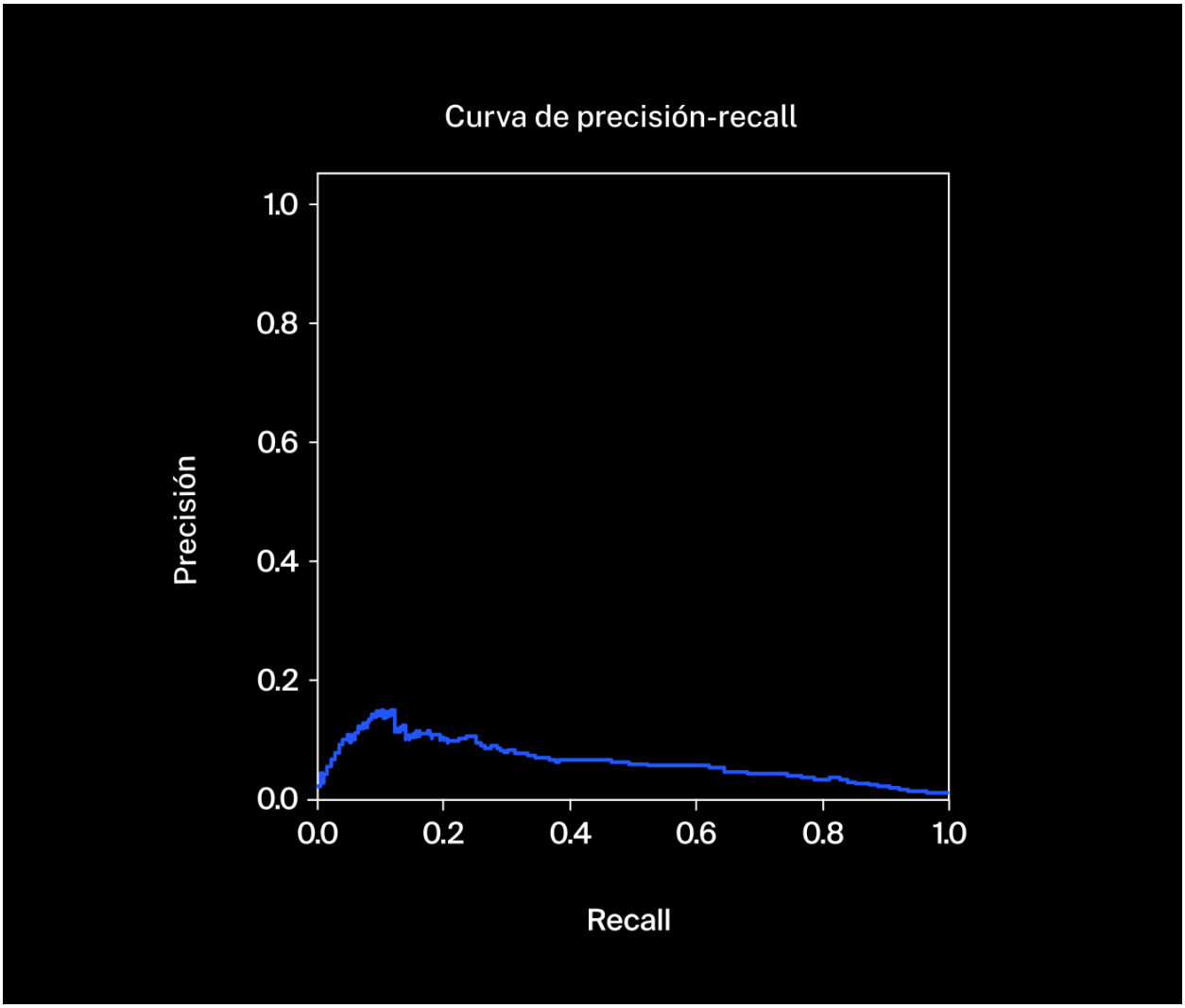
```
for value in np.arange(first, last, step):
```

Curva PR

Tracemos los valores de las métricas y veamos cómo responde la curva al cambio de umbral.

En la gráfica, el valor de precisión se traza verticalmente y recall, horizontalmente. Una curva trazada a partir de los valores de Precisión y Recall se denomina curva PR.

Cuanto más alta sea la curva, mejor será el modelo.



TVP y TFP

No puedes calcular la precisión cuando no hay observaciones positivas. Consideremos las métricas que no implican división entre cero.

Antes de pasar a la nueva curva, definamos algunos términos importantes.

Tasa de verdaderos positivos (True Positive Rate), o *TVP*, es el resultado de dividir las respuestas *VP* entre todas las respuestas positivas. Aquí está la fórmula, donde P es todas las respuestas positivas:

$$TVP = \frac{VP}{P}$$

La tasa de falsos positivos, o TFP, es el resultado de dividir las respuestas FP entre todas las respuestas negativas. Se calcula utilizando una fórmula similar, donde N son todas las respuestas negativas:

$$TFP = \frac{FP}{N}$$

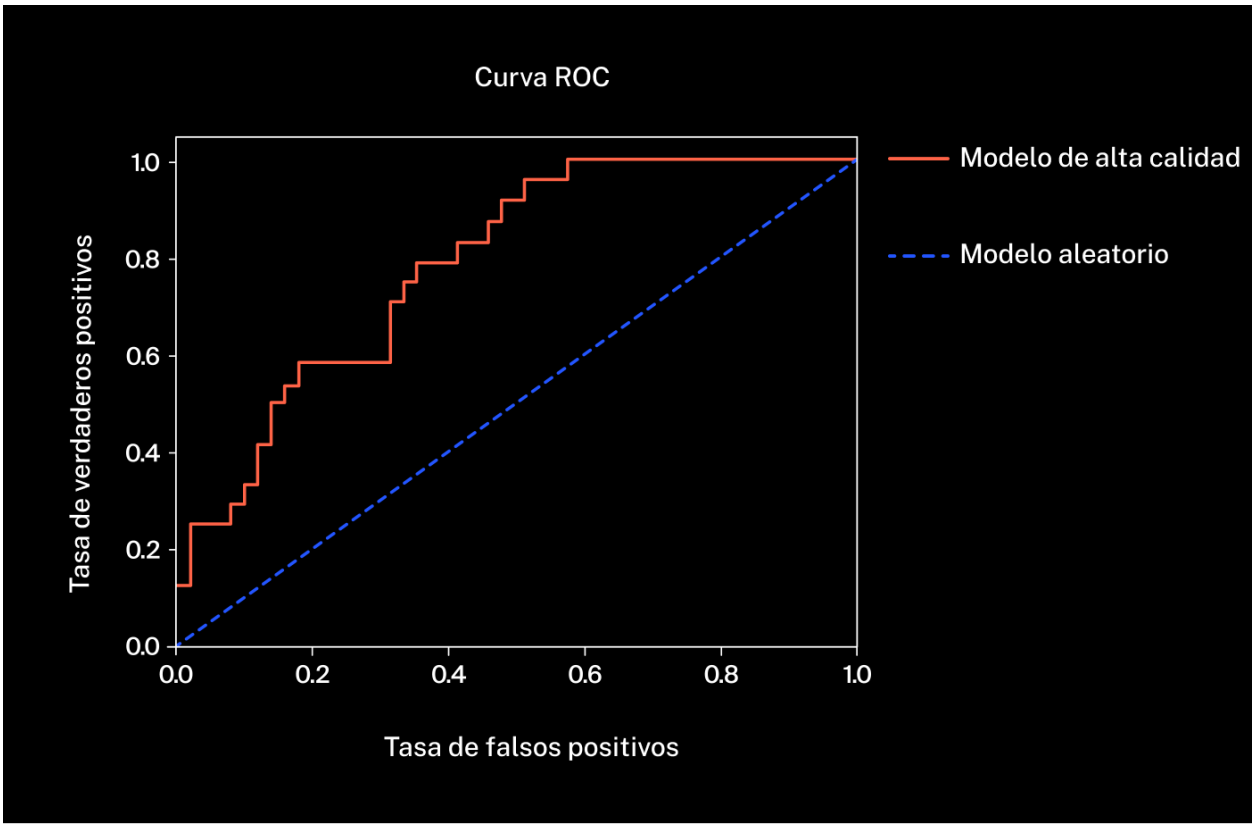
No habrá división entre cero: los denominadores son valores constantes que no dependen de cambios en el modelo.

Curva ROC

Hemos visto un nuevo enfrentamiento: TVP vs. TFP. Tracemos la curva.

Colocamos los valores de la tasa de falsos positivos (TFP) a lo largo del eje horizontal y los valores de la tasa de verdaderos positivos (TVP) a lo largo del eje vertical. Luego iteramos los valores del umbral de regresión logística y trazamos una curva. Se denomina **curva ROC** (***R****eceiver* ***O****perating* ***C****haracteristic, un término de la teoría del procesamiento de señales*).

Para un modelo que siempre responde de forma aleatoria, la curva *ROC* es una línea diagonal que va desde la esquina inferior izquierda hasta la esquina superior derecha. Cuanto más alta sea la curva, mayor será el valor de *TVP* y mejor será la calidad del modelo.



Para encontrar cuánto difiere nuestro modelo del modelo aleatorio, calculemos el valor **AUC-ROC** (**A**rea **U**nder **C**urve ROC o área bajo la **curva ROC).** Esta es una nueva métrica de evaluación con valores en el rango de 0 a 1. El valor AUC-ROC para un modelo aleatorio es 0.5.

Podemos trazar una curva ROC con la variable *roc_curve()* del módulo *sklearn.metrics*:

```
from sklearn.metrics import roc_curve
```

Esta toma los valores objetivo y las probabilidades de clase positivas, supera diferentes umbrales y devuelve tres listas: valores de *TFP*, valores de *TVP*, y los umbrales que superó.

```
fpr, tpr, thresholds = roc_curve(target, probabilities)
```

Para calcular *AUC-ROC*, usa la función *roc_auc_score()* de la librería *sklearn*:

```
from sklearn.metrics import roc_auc_score
```

A diferencia de otras métricas, esta toma probabilidades de clase "1" en lugar de predicciones:

```
auc_roc = roc_auc_score(target_valid, probabilities_one_valid)
```