Sprint 9

Capítulo 1/8

Introducción al curso

**1. Aprendizaje supervisado**

Sigamos explorando el aprendizaje supervisado.

Ya sabes sobre las características y el objetivo, puedes distinguir la clasificación de la regresión y puedes evaluar la calidad de un modelo.

En este curso aprenderás nuevas métricas y los casos en los que puedes usarlas. Aprenderás a ajustar los modelos de machine learning y mejorar las métricas.

**Estructura del curso**

Tres capítulos de este curso están dedicados a la clasificación y el último se centra en la regresión.

En los primeros tres capítulos aprenderemos a preparar características para el análisis mediante la transformación de características categóricas y la mejora de características numéricas.

Luego construiremos un modelo y veremos si la métrica de *exactitud* es apropiada para su evaluación. También trabajaremos con nuevas métricas e intentaremos mejorarlas para resolver tareas de clasificación desequilibradas.

En el último capítulo repasaremos diferentes métricas de regresión y sus aplicaciones.

Al final del curso trabajarás en tu propio proyecto. Será un problema de clasificación donde tendrás que predecir el abandono de clientes (el porcentaje de la base de clientes de una empresa que deja de usar sus servicios por período de tiempo) para un banco. Haz clic [aquí](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_8/moved_DS_ESP_Descripcin_del_proyecto.pdf) para ver la descripción del proyecto y lo que implica.

**Tus objetivos:**

* Aprender nuevas métricas de evaluación para la clasificación: precisión, recall (sensibilidad), valor F1 y AUC-ROC. Calcular sus valores utilizando las herramientas scikit-learn.
* Resolver tareas de clasificación desequilibradas. Eso incluye ajustar los pesos de las clases y sus umbrales.
* Aprender las nuevas métricas de evaluación para regresión: *EAM* y *R²*. Calcular sus valores utilizando las herramientas *scikit-learn*.

Tu enfoque para este sprint es:

Código QR

Descripción generada automáticamente

Logotipo

Descripción generada automáticamente

Código QR

Descripción generada automáticamente

**¿Cuánto tiempo llevará?**

Este va a ser un curso difícil, con muchas fórmulas, cálculos y ejercicios prácticos. ¡Así que abróchate el cinturón y comencemos! Espera pasar entre 30 y 50 horas para completar este material, dependiendo de tus conocimientos previos y hábitos de estudio. Si sientes que te estás quedando atrás, no dudes en ponerte en contacto con nuestro equipo de orientación para informarles de esto. Como siempre, nos comprometemos a ayudarte en cada paso del camino.

Codificación y estandarización de datos

**Introducción**

**Para las tareas de machine learning no solo necesitamos preparar datos, sino también características.**

¿Qué aprenderás?

* Transformar características categóricas con codificación One-Hot para entrenar una regresión logística.
* Realizar la codificación ordinal de características categóricas para entrenar un árbol de decisión.
* Mejorar las métricas de evaluación escalando características.

**¿Cuánto tiempo tomará?**

Diez lecciones de 5-10 minutos cada una.

**Descripción del ejercicio**

Prepararemos datos para entrenar un modelo que prediga si un cliente hará un reclamo de seguro.

**Carga de datos**

**Analicemos el ejercicio, carguemos los datos e investiguemos las características.**

Los ingresos de los pagos de la clientela superan con creces los costos de liquidación de reclamaciones. El plan de cada compañía de seguros es que solo una pequeña parte de la clientela hará un reclamo, mientras que la mayoría de las personas no recurrirán a él.

Cada cliente que se une es una caja negra para la compañía de seguros de viaje. ¿Qué probabilidad hay de que la empresa tenga que pagar? Históricamente, solo el 1 % de la clientela reclama beneficios de seguro. ¿Cómo predecir, entonces, si una persona lo necesitará?

El machine learning nos permite calcular la probabilidad de una reclamación de seguro.



Tenemos las siguientes características:

* Agency: nombre de la agencia de seguros
* Agency Type: tipo de agencia de seguros
* Distribution Channel*:* canal de distribución de la agencia de seguros
* Product Name: nombre del producto de seguro
* Duration: duración del viaje (días)
* Destination: destino del viaje
* Net Sales: ventas netas ($)
* Commission: comisión de la agencia de seguros ($)
* Gender: género de la persona asegurada
* Age: edad de la persona asegurada

Objetivo:

* Claim — reclamación de liquidación (1 es sí, 0 es no)
* 1.
* Carga los datos de /datasets/travel\_insurance\_us.csv a la variable data. Muestra en la pantalla los primeros diez elementos. Mira los datos.

import pandas as pd

data = pd.read\_csv("/datasets/travel\_insurance\_us.csv")# < escribe tu código aquí >

print(data.head(1))

Resultado

Agency Agency Type Distribution Channel ... Commission (in value) Gender Age

0 JZI Airlines Online ... 15.75 M 39

1 EPX Travel Agency Online ... 0.00 None 36

2 EPX Travel Agency Online ... 0.00 None 36

3 EPX Travel Agency Online ... 0.00 None 36

4 JZI Airlines Online ... 15.75 M 34

5 EPX Travel Agency Online ... 0.00 None 36

6 C2B Airlines Online ... 36.00 F 37

7 EPX Travel Agency Online ... 0.00 None 26

8 EPX Travel Agency Online ... 0.00 None 36

9 EPX Travel Agency Online ... 0.00 None 35

[10 rows x 11 columns]

¡Es correcto!

Lo que obtienes:

* Muchas de las características son categóricas (por ejemplo, Agency)
* Faltan valores de Gender.

El objetivo Сlaim es 0 para toda la clientela en la parte superior de nuestra tabla. ¡Sí, regresaron a casa a salvo!

2.

Divide los datos en dos conjuntos:

* conjunto de entrenamiento (train)
* conjunto de validación (valid) — 25% de los datos de origen

Especifica random\_state=12345. Declara cuatro variables y almacena las características y el objetivo de la siguiente manera:

* características: features\_train, features\_valid
* objetivo: target\_train, target\_valid

Muestra en pantalla los tamaños de las tablas almacenadas en las variables: features\_train y features\_valid.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv('/datasets/travel\_insurance\_us.csv')

# Dividir los datos en características y la variable objetivo

train, valid = train\_test\_split(data, test\_size=0.25, random\_state=12345)

# Separar las características y la variable objetivo para el conjunto de entrenamiento

features\_train = train.drop('Claim', axis=1)

target\_train = train['Claim']

# Separar las características y la variable objetivo para el conjunto de validación

features\_valid = valid.drop('Claim', axis=1)

target\_valid = valid['Claim']

# < escribe tu código aquí >

print(features\_train.shape)

print(features\_valid.shape)

Resultado

(37995, 10)

(12665, 10)

¡Es correcto!

Vaya, eso es una gran variedad de datos de entrenamiento, como en un bufete donde puedes comer todo lo que quieras. Solo asegúrate de no comer en exceso. Es posible que el seguro no lo cubra.

**Entrenamiento de pilotos**

**Para predecir la clase, necesitamos a nuestra vieja amiga, la regresión logística.**

La regresión logística se utiliza para tareas de clasificación. En nuestro caso, es binaria: se presenta el reclamo o no.

Intentaremos entrenar nuestro modelo. ¿Crees que podemos entrenarlo usando los datos sin procesar? ¡Vamos a arriesgarnos!

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

data = pd.read\_csv('travel\_insurance\_us.csv')

train, valid = train\_test\_split(data, test\_size=0.25, random\_state=12345)

features\_train = train.drop('Claim', axis=1)

target\_train = train['Claim']

features\_valid = valid.drop('Claim', axis=1)

target\_valid = valid['Claim']

model = LogisticRegression()

model.fit(features\_train, target\_train)

...

ValueError: could not convert string to float: 'M'

Muy bien, ¡adelante! Es el momento perfecto para un "te lo dije". Tenías razón, cometimos un error.

¿Qué salió mal?

La regresión logística determina la categoría utilizando una fórmula que consta de características numéricas. Además de numéricas, nuestros datos contenían características categóricas, de ahí el error.

Verifica los tipos de características almacenadas en la tabla. Muéstralos. Luego muestra en pantalla los primeros cinco valores de la columna Gender

mport pandas as pd

data = pd.read\_csv('/datasets/travel\_insurance\_us.csv')

print(data.dtypes)

print(data['Gender'].head(5))

# < escribe tu código aquí >

Resultado

Agency object

Agency Type object

Distribution Channel object

Product Name object

Claim int64

Duration int64

Destination object

Net Sales float64

Commission (in value) float64

Gender object

Age int64

dtype: object

0 M

1 None

2 None

3 None

4 M

Name: Gender, dtype: object

¡Es correcto!

Encontraste objetos sospechosos. Paso 1: Haz una cara sospechosa. Paso 2: Convierte objetos sospechosos en algo sencillo. Te diremos cómo en la siguiente lección.

# **Codificación One-Hot**

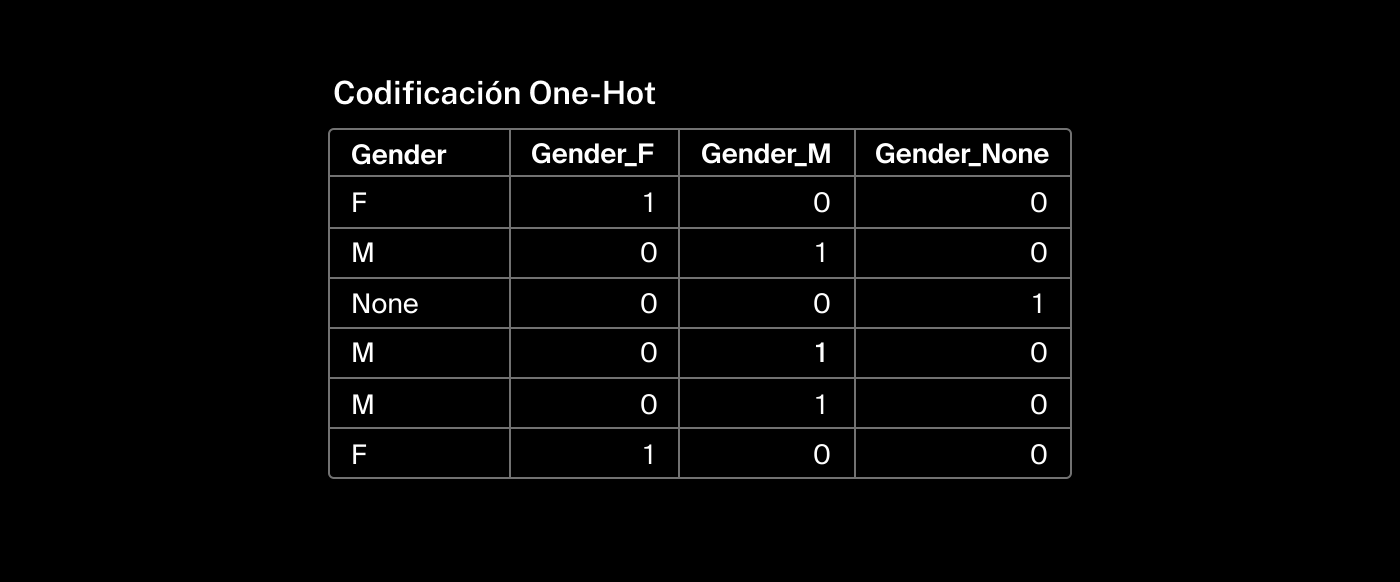
**Existe una técnica especial para transformar características categóricas en numéricas. Se llama**One-Hot Encoding (OHE)**.**

¿Cómo funciona *One-Hot Encoding*? Vamos a utilizar los valores característicos de *Gender*

Vamos a crear una columna separada para cada valor de Gender (F, M, Ninguno):

* Gender\_F
* Gender\_M
* Gender\_None

La columna que obtiene un 1 depende del valor de la característica "Gender". Si el valor es F, 1 va a la columna Gender\_F; si es M, entonces 1 va a Gender\_M.



Recapitulemos. La técnica *OHE* nos permite transformar características categóricas en características numéricas en dos pasos:

1. Agrega una columna separada para cada valor de función.
2. Si la categoría se ajusta a la observación, se asigna 1, de lo contrario, se asigna 0.

Las nuevas columnas (Gender\_F, Gender\_M, Gender\_None) se denominan variables dummy.

La librería pandas tiene una función pd.get\_dummies() que se puede usar para obtener variables dummy.

Podemos mirar los valores de la columna Gender usando *OHE*. Llama a pd.get\_dummies() y muestra en pantalla las primeras cinco filas de la tabla transformada.  
  
import pandas as pd

data = pd.read\_csv('/datasets/travel\_insurance\_us.csv')

print(pd.get\_dummies(data['Gender']).head(5))  
  
esultado

F M None

0 0 1 0

1 0 0 1

2 0 0 1

3 0 0 1

4 0 1 0

¡Es correcto!

Como puedes ver, se necesitan tres dummies para encontrar dos hombres.

# La trampa dummy

### Como es de esperar, en One-Hot Encoding hay más de lo que parece. Cuando los datos son abundantes, tenemos la posibilidad de caer en la trampa de las características dummy. Aquí aprenderemos cómo evitarla.

Para obtener tu identificación del Departamento de Vehículos Motorizados, debes presentar un comprobante de residencia actual en tu estado. Tú sabes cómo es el departamento, así que para no fallar y terminar en una sola visita, traes la factura de servicios públicos, el contrato de hipoteca y la factura del impuesto a la propiedad, aunque sería suficiente traer dos de esos documentos. Si bien traer todos los documentos que puedas para luchar contra la burocracia puede ser una buena idea al tratar con el deparmento, no se aplica el mismo principio a la capacitación de modelos. Si mantienes las características como están ahora, esto dificultará el proceso de entrenamiento.

Hemos agregado tres columnas nuevas a nuestra tabla, pero su alta correlación confundirá a nuestro modelo. Para evitar esto, podemos eliminar con seguridad cualquier columna, ya que sus valores se pueden deducir fácilmente de las otras dos columnas (tiene 1 donde las otras dos columnas tienen ceros y tiene ceros en el resto). De esta manera, no caeremos en la trampa dummy.

Para eliminar la columna, llama a la función pd.get\_dummies() junto con el parámetro drop\_first. Si pasas drop\_first=True entonces se elimina la primera columna. De lo contrario, es drop\_first=False por defecto y no se descartan columnas.

1.

Programa la variable**Gender** con OHE. Llama a pd.get\_dummies() con el argumento drop\_first para evitar la trampa dummy. Muestra en pantalla las primeras cinco filas de la tabla modificada.

import pandas as pd

data = pd.read\_csv('/datasets/travel\_insurance\_us.csv')

print(pd.get\_dummies(data['Gender'], drop\_first=True).head(5))

# < escribe el código aquí >

Resultado

M None

0 1 0

1 0 1

2 0 1

3 0 1

4 1 0

# La trampa dummy

### Como es de esperar, en One-Hot Encoding hay más de lo que parece. Cuando los datos son abundantes, tenemos la posibilidad de caer en la trampa de las características dummy. Aquí aprenderemos cómo evitarla.

Para obtener tu identificación del Departamento de Vehículos Motorizados, debes presentar un comprobante de residencia actual en tu estado. Tú sabes cómo es el departamento, así que para no fallar y terminar en una sola visita, traes la factura de servicios públicos, el contrato de hipoteca y la factura del impuesto a la propiedad, aunque sería suficiente traer dos de esos documentos. Si bien traer todos los documentos que puedas para luchar contra la burocracia puede ser una buena idea al tratar con el deparmento, no se aplica el mismo principio a la capacitación de modelos. Si mantienes las características como están ahora, esto dificultará el proceso de entrenamiento.

Hemos agregado tres columnas nuevas a nuestra tabla, pero su alta correlación confundirá a nuestro modelo. Para evitar esto, podemos eliminar con seguridad cualquier columna, ya que sus valores se pueden deducir fácilmente de las otras dos columnas (tiene 1 donde las otras dos columnas tienen ceros y tiene ceros en el resto). De esta manera, no caeremos en la trampa dummy.

Para eliminar la columna, llama a la función pd.get\_dummies() junto con el parámetro drop\_first. Si pasas drop\_first=True entonces se elimina la primera columna. De lo contrario, es drop\_first=False por defecto y no se descartan columnas.

1.

Programa la variable**Gender** con OHE. Llama a pd.get\_dummies() con el argumento drop\_first para evitar la trampa dummy. Muestra en pantalla las primeras cinco filas de la tabla modificada.

import pandas as pd

data = pd.read\_csv('/datasets/travel\_insurance\_us.csv')

print(pd.get\_dummies(data['Gender'], drop\_first=True).head(5))

# < escribe el código aquí >  
Resultado

M None

0 1 0

1 0 1

2 0 1

3 0 1

4 1 0

¡Es correcto!

Resulta que no se necesitan tres dummies para encontrar dos hombres.

2.

Programa todo el DataFrame con One-Hot. Llama a pd.get\_dummies() con el argumento drop\_first. Almacena la tabla en la variable data\_ohe.

Muestra en pantalla las primeras tres filas de la tabla resultante.

import pandas as pd

data = pd.read\_csv('/datasets/travel\_insurance\_us.csv')

data\_ohe = pd.get\_dummies(data, drop\_first=True)

print(data\_ohe.head(3))# < escribe el código aquí >  
  
Resultado

Claim Duration Net Sales ... Destination\_ZIMBABWE Gender\_M Gender\_None

0 0 12 45.0 ... 0 1 0

1 0 50 22.0 ... 0 0 1

2 0 251 80.0 ... 0 0 1

[3 rows x 189 columns]

¡Es correcto!

La tabla se hizo más grande. Ahora tienes nuevas columnas para algunos valores de características: nombres de agencias, destinos, Gender\_M y Gender\_None. La columna Gender fue eliminada. Así es como funciona OHE.

3.

Divide los datos de origen en dos conjuntos utilizando la proporción de 75:25 (%):

* entrenamiento (train)
* validación (valid)

Declara cuatro variables y almacena las características y el objetivo de la siguiente manera:

* características: features\_train, features\_valid
* objetivo: target\_train, target\_valid

Vas a dominar una forma alternativa de usar la función train\_test\_split(): puede tomar dos variables (características y objetivo). Consulta la documentación de este método en particular.

Entrena una regresión logística. Muestra en pantalla el texto "¡Entrenado!" (ya en precódigo) para asegurarte de que el código terminó de ejecutarse sin problemas.

Especifica random\_state=12345 tanto para la división de datos como para el entrenamiento del modelo.

Al entrenar una regresión logística, es posible que encuentres una advertencia de sklearn. Esta no te impedirá pasar la tarea, pero si no te gusta ver un montón de código en rojo en tu pantalla, especifica el argumento solver='liblinear'; librería linear:

model = LogisticRegression(solver='liblinear')

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

data = pd.read\_csv('/datasets/travel\_insurance\_us.csv')

data\_ohe = pd.get\_dummies(data, drop\_first=True)

target = data\_ohe['Claim']

features = data\_ohe.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(features, target, test\_size=0.25, random\_state=12345)

# < escribe el código aquí >

model = LogisticRegression(solver='liblinear',random\_state=12345)

model.fit(features\_train, target\_train)

print('¡Entrenado!')  
  
Resultado

¡Entrenado!

¡Es correcto!

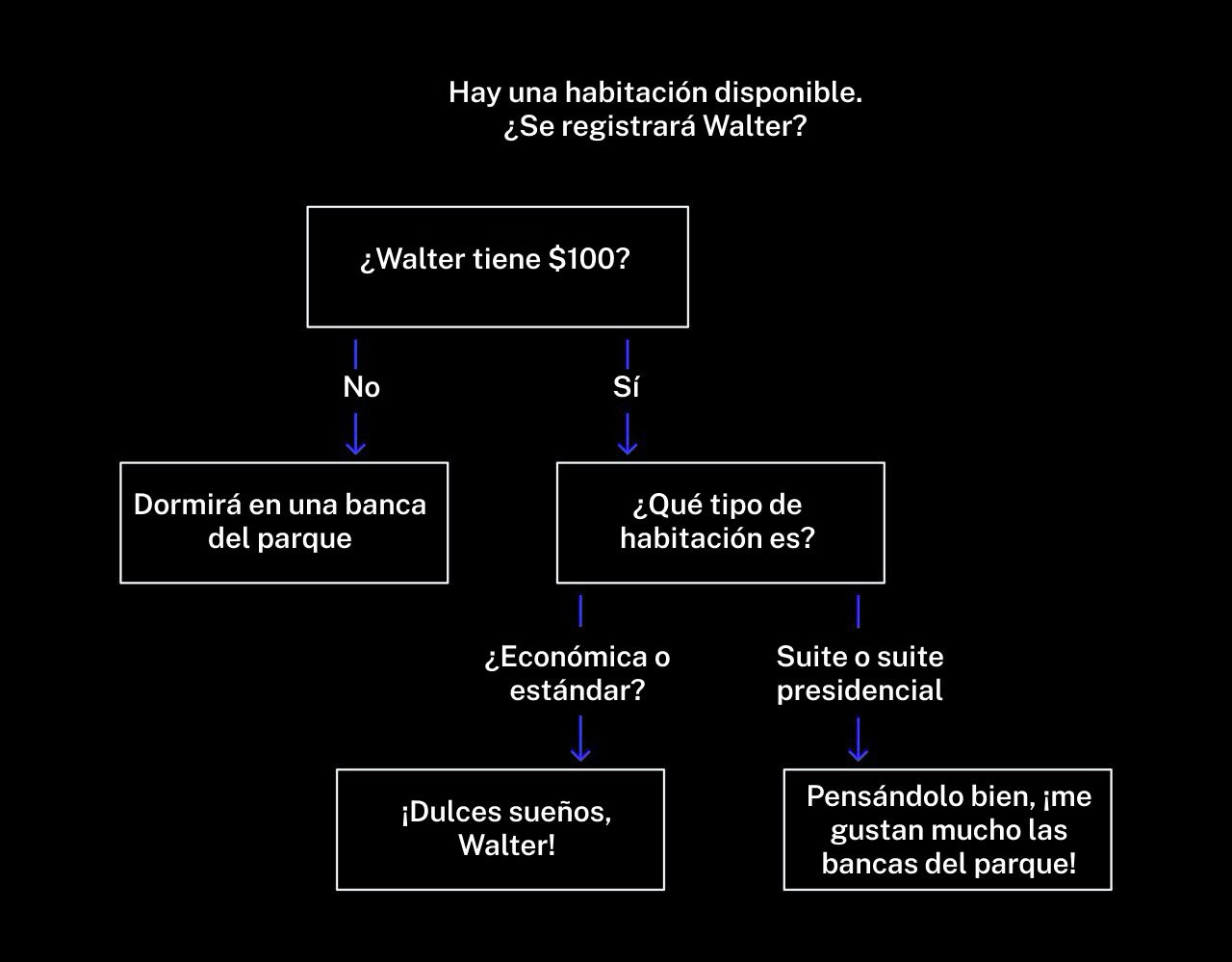
En esta lección, mencionamos 12 veces la palabra dummies. Son suficientes por un día.Sigamos adelante.

# Codificación de etiquetas

### Echemos un vistazo a una nueva técnica de codificación de características para árboles de decisión y bosques aleatorios.

El árbol de decisión hace preguntas a los nodos, así que tal vez no necesite ningún tipo de codificación y pueda trabajar con características categóricas directamente.

Así es como se ve como un diagrama:



Intentemos entrenar el modelo:

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

data = pd.read\_csv('travel\_insurance\_us.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345)

tree = DecisionTreeClassifier(random\_state=12345)

tree.fit(features\_train, target\_train)

...

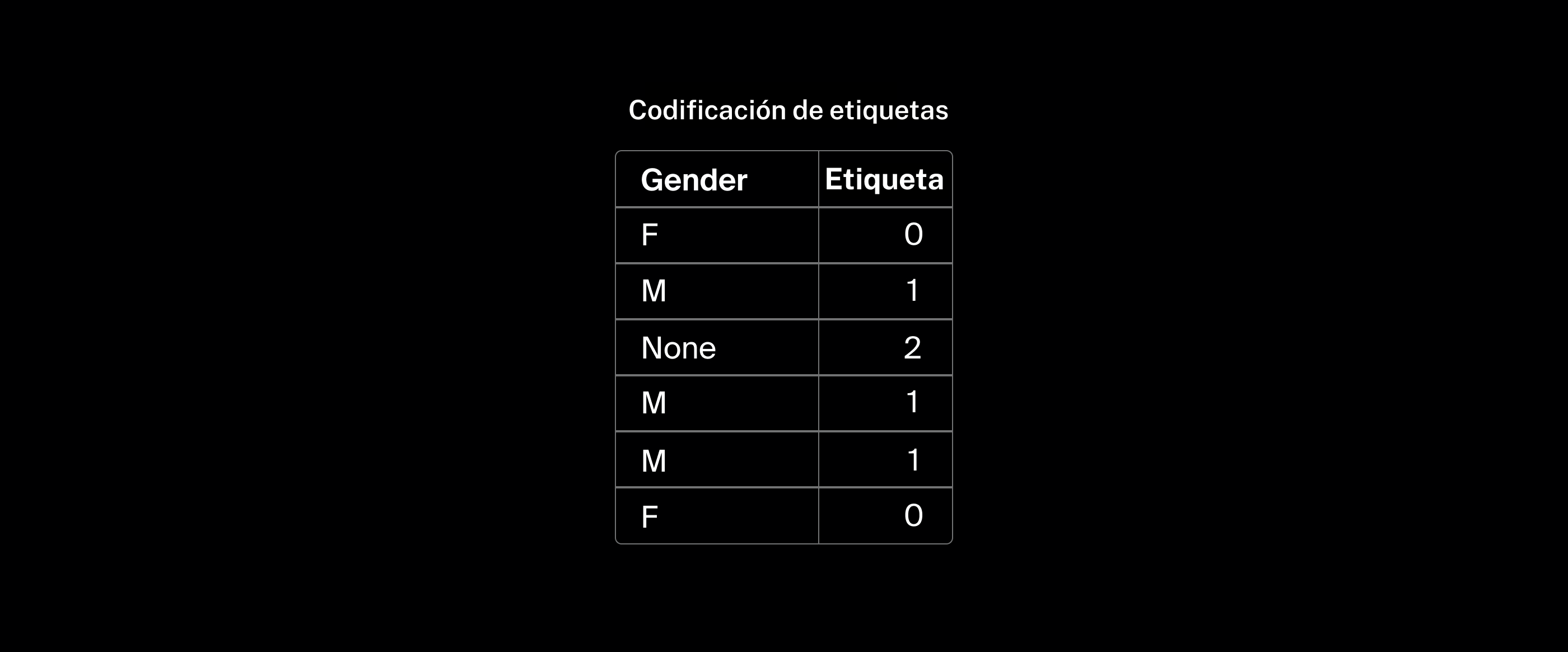
ValueError: could not convert string to float: 'EPX'

Hay otro error. Esta vez no fue causado por el modelo subyacente en sí mismo que exige solo valores numéricos, sino por cómo se implementan los árboles en la librería *sklearn*. Algunas otras plataformas de machine learning, como R o H2O, pueden aprovechar los datos categóricos sin procesar. Desafortunadamente, en este momento, *sklearn* no es compatible con esta característica y requiere codificación.

Entonces, ¿se supone que debemos usar OHE para los árboles de decisión también? Esta podría ser una opción viable, pero no es la mejor por varias razones. La razón principal es que, a diferencia de los modelos de regresión que tienen acceso a todo el espectro de variables al mismo tiempo, el modelo basado en árboles solo puede procesar una variable a la vez. Como resultado, no tiene información completa sobre la variable categórica original. En cambio, su importancia se dispersa entre muchas variables dummy, por lo que estas variables dummy casi nunca se eligen como variables de división cerca de la raíz del árbol, incluso si fueran un buen predictor.

Parece que necesitamos una nueva técnica de codificación, una que mantenga toda la información sobre la variable original en una característica. La solución obvia es simplemente reemplazar las categorías con etiquetas numéricas arbitrarias: codificación de etiquetas (label encoding).

Esta técnica se utiliza para codificar características categóricas para árboles de decisión y otros algoritmos basados en árboles, como bosques aleatorios.



*Sklearn* proporciona dos clases para la codificación de etiquetas y ambas se encuentran en el módulo sklearn.preprocessing. La clase LabelEncoder se usa para codificar una sola columna, mientras que la clase OrdinalEncoder se usa para dos o más columnas a la vez, hasta el conjunto de datos completo. Esta es la principal diferencia entre ellas y, por lo demás, funcionan prácticamente de la misma manera, asignando números a categorías en orden alfabético (las etiquetas numéricas exactas asignadas no importan y también podrían ser aleatorias, por lo que el orden alfabético es un enfoque aceptable).

Es importante tener en cuenta que actualmente existe cierta confusión en la comunidad de Data Science con respecto a la terminología de esta codificación. Dado que la clase *OrdinalEncoder* de la librería sklearn generalmente se usa para fines de codificación de etiquetas, es posible que encuentres fuentes que la llamen Сodificación Ordinal. Sin embargo, estrictamente hablando, la codificación ordinal es un tipo diferente de codificación que trataremos por separado en la siguiente lección.

En aras de la simplicidad, solo consideraremos el uso de *OrdinalEncoder* en todo el conjunto de datos y solo para fines de codificación de etiquetas.

Importa *OrdinalEncoder* de la librería:

from sklearn.preprocessing import OrdinalEncoder

La transformación se realiza en tres pasos:

1. Crea una instancia de esta clase.

encoder = OrdinalEncoder()

1. Antes de usar el codificador, debes ajustarlo a tus datos para que reconozca qué variables son categóricas. Para hacerlo, llama al método fit() y pásale los datos (de la misma manera que lo hacemos cuando entrenamos un modelo).

encoder.fit(data)

1. Utiliza el método **transform()**. Los datos transformados se almacenarán en la variable **data\_ordinal** .

data\_ordinal = encoder.transform(data)

Usa el constructor DataFrame() para agregar nombres de columna*:*

data\_ordinal = pd.DataFrame(encoder.transform(data),

columns=data.columns)

Si necesitas transformar los datos solo una vez, como en nuestro caso, puedes llamar al método **fit\_transform()** en su lugar. Este combina fit() y transform()*.*

data\_ordinal = pd.DataFrame(encoder.fit\_transform(data),

1.

Transforma las funciones utilizando *codificación de etiquetas.* Importa *OrdinalEncoder* desde el módulo *sklearn.preprocessing*.

Almacena el resultado en la variable data\_ordinal. Utiliza el constructor *DataFrame()*.

Muestra en pantalla las primeras cinco filas (en precódigo).

2.

Utiliza los datos transformados para entrenar un árbol de decisión. Muestra en pantalla el texto "¡Entrenado!" (en precódigo) para asegurarte de que el código completó su ejecución.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.preprocessing import OrdinalEncoder

data = pd.read\_csv('/datasets/travel\_insurance\_us.csv')

encoder = OrdinalEncoder()

data\_ordinal = pd.DataFrame(encoder.fit\_transform(data), columns=data.columns)

target = data\_ordinal['Claim']

features = data\_ordinal.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = DecisionTreeClassifier(random\_state=12345)

model.fit(features\_train, target\_train)

# < escribe el código aquí >

print('¡Entrenado!')  
  
Resultado

¡Entrenado!

¡Es correcto!

Nuestro modelo está entrenado, ¡y tú mandas! Pon orden en las categorías salvajes.

Capítulo 2/8

Codificación y estandarización de datos

# Codificación ordinal

### En la lección anterior mencionamos que la codificación ordinal difiere de la de etiquetas. ¿Cuál es la diferencia?

Las variables categóricas que cubrimos hasta ahora no tenían ningún tipo de orden natural implícito. Para categorías como "Plan de valor", "Plan de plata" y "Plan de cancelación", no había una forma obvia de decidir qué categoría se etiquetaba como 0, 1 o 2. Así que simplemente asignamos estas etiquetas de forma arbitraria.

Sin embargo, hacerlo no siempre es una buena idea. Imagina una variable que contiene descripciones de evaluación de calidad, como "excelente", "buena" y "mala". O quizás una variable que contenga descripciones de temperatura, como "caliente", "tibia" y "fría". Este tipo de variables categóricas tienen un orden natural, por lo que asignarles etiquetas de forma arbitraria sería un error. Si codificas "fría" con 0, "caliente" con 1 y "tibia" con 2, entonces el algoritmo funcionará suponiendo que "caliente" es una cualidad entre "fría" y "tibia", en lugar de una cualidad aún más cálida que "tibia".

Este tipo de variable categórica se denomina variable ordinal, a diferencia de una variable nominal (una variable de categorías sin orden). La codificación ordinal es una codificación de una variable ordinal con etiquetas numéricas dispuestas en un orden natural específico, generalmente realizada mediante enumeración manual de etiquetas.

Es técnicamente posible implementar la codificación ordinal mediante la clase *OrdinalEncoder* en *sklearn*. Para hacerlo, debes especificar el parámetro *categories*. Sin embargo, la clase *OrdinalEncoder* de *sklearn* no es fácil de usar cuando se trata de codificación ordinal, por lo que sugerimos usar una clase *OrdinalEncoder* alternativa de la librería *category\_encoders*. Su parámetro *mapping* está mejor documentado y es mucho más intuitivo de usar. Otra alternativa es simplemente implementar el mapeo por medio de *pandas*. Ya que estamos usando *pandas* de todos modos, este es el método que recomendamos:

temperature\_dict = {'cold': 0, 'warm': 1, 'hot': 2}

df['temperature'] = df['temperature'].map(temp\_dict)

Capítulo 2/8

Codificación y estandarización de datos

# Descripción general de la codificación

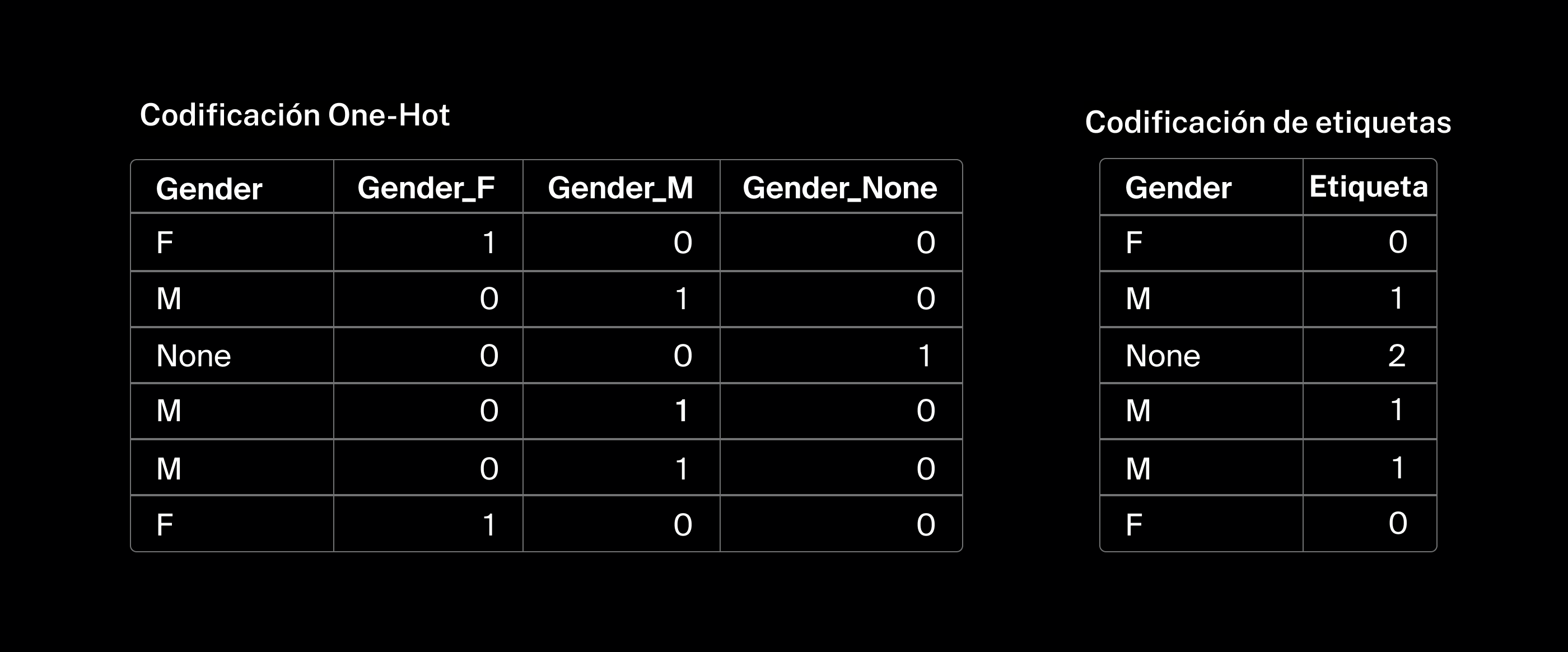
### Averigüemos cómo elegir el método de codificación correcto y por qué la codificación de etiquetas no es adecuada para la regresión logística.

En términos generales, la codificación de etiquetas es una mala idea, especialmente para los algoritmos de regresión. Dichos algoritmos se basan en ponderaciones de aprendizaje para cada característica en una fórmula. Sin embargo, si codificamos una variable categórica con números en una sola columna, entonces el algoritmo tratará la variable como continua y asumirá que los valores están en una escala significativa. Si "gato" es 0 y "perro" es 1, no queremos que nuestro modelo piense que "perro" es de alguna manera mayor que "gato", o que operaciones aritméticas como "(gato + perro) / 2 = medio perro" tengan sentido. Tampoco queremos que el modelo use el mismo peso para varias entidades completamente diferentes.

Entonces, en caso de regresión, las categorías deben codificarse de una manera que les asigne la misma importancia a todas, pero que aún así se reconozcan como distintas y diferentes. Es por eso que OHE (One-Hot Encoding) es necesario en primer lugar. OHE codifica cada categoría como un vector linealmente independiente en un espacio N-dimensional. Todos estos vectores son inicialmente equidistantes entre sí y aprenderán sus pesos independientes separados durante la ejecución del algoritmo. Es por eso que OHE funciona bien para todos los algoritmos de machine learning que analizan todas las funciones al mismo tiempo durante el entrenamiento.

Para los algoritmos basados en árboles, OHE no es deseable por las razones que ya explicamos en una lección anterior. Todavía se puede usar, pero si se usa, es probable que las variables codificadas mediante OHE se ignoren en gran medida. El mejor enfoque para los árboles es hacer divisiones por categorías sin procesar, pero dado que *scikit-learn* actualmente no admite esta opción, en muchos casos, el segundo mejor enfoque resulta ser la codificación de etiquetas.

Los inconvenientes de la codificación de etiquetas se ven mitigados por el hecho de que para los árboles, aunque el orden de las etiquetas sigue siendo importante, la escala de las etiquetas no lo es. En otras palabras, no hay una diferencia real entre las etiquetas [0, 1, 2, 3] y las etiquetas [0, 2, 5, 40]. Y gracias a eso, a pesar de que la codificación de etiquetas todavía crea información que no refleja ningún hecho real y, a veces, da como resultado divisiones que no tienen ningún sentido, las aplicaciones prácticas muestran que con frecuencia da como resultado divisiones útiles, especialmente cuando se experimenta con diferentes ordenamientos de las etiquetas. Además, algunos frameworks, como LightGBM, aún pueden tratar las etiquetas numéricas como categóricas mediante configuraciones específicas, eliminando cualquier complicación relacionada con la codificación de etiquetas.



Todo lo anterior se aplica a las variables nominales. En cuanto a las variables ordinales, se recomienda utilizar la codificación ordinal para ellas, independientemente del algoritmo. A diferencia de las variables nominales, que categorizan sin un orden inherente, las variables ordinales sí representan un orden o secuencia. Cada categoría de una variable ordinal refleja un nivel o magnitud en una escala, aunque esa escala no se mida numéricamente. Un ejemplo común es la medición de la satisfacción o la intensidad del dolor, donde 'no se puede medir la felicidad' directamente, pero sí se pueden ordenar los niveles de felicidad percibida de menor a mayor.

Por lo tanto, es razonable suponer que una variable ordinal tenga un solo peso en la regresión. Si la propiedad de ser "fría" es importante para la predicción, entonces cualquier otra temperatura es igualmente importante. Es por eso que representarlos como números en una sola columna por lo general funciona mejor que OHE, siempre que el orden de las etiquetas sea correcto.

Finalmente, también es importante señalar que cuantas más categorías tenga que codificar la variable, mayores serán los inconvenientes de OHE. Las variables que tienen muchas categorías se denominan variables de alta cardinalidad, y el uso de OHE para codificarlas conduce a un alto consumo de memoria, escasez de datos y otros problemas típicamente asociados con la alta dimensionalidad de los datos. Si bien todavía se puede usar OHE, es probable que las técnicas de codificación más avanzadas (codificación binaria, hashing, codificadores bayesianos, etc.) brinden mejores resultados. Sin embargo, no los cubriremos porque, por el momento, OHE será suficiente.

Para resumir todo, la codificación de una variable categórica con OHE generalmente dará como resultado el mejor rendimiento a menos que:

1. Se use en un algoritmo basado en árboles (algunas plataformas de machine learning admiten variables categóricas no codificadas para árboles, pero si no, la codificación de etiquetas puede dar como resultado un mejor rendimiento en algunos casos).
2. Sea una variable ordinal (la codificación ordinal generalmente funciona mejor para las variables ordinales).
3. Sea una variable de alta cardinalidad (es posible que se necesiten técnicas de codificación más avanzadas).

De ahora en adelante, usaremos OHE solo para codificar variables categóricas. Nuestro conjunto de datos no contiene ninguna variable ordinal que requiera codificación ordinal, y evitaremos usar la codificación de etiquetas para simplificar. OHE es una técnica de codificación universal que nos dará resultados decentes aunque no es óptima para los árboles.

# Escalado de características

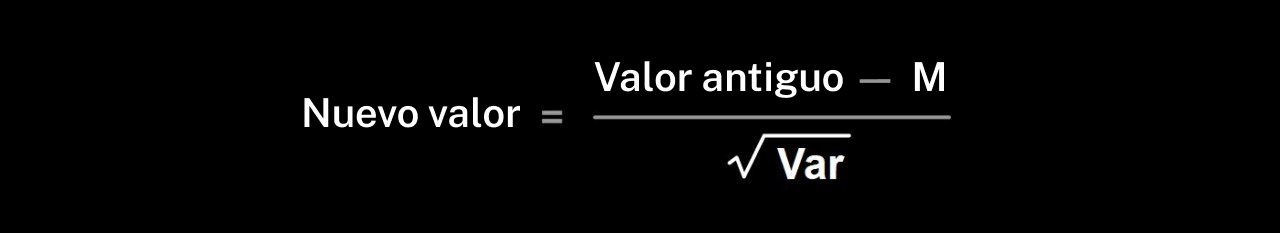
### ¿Qué debemos hacer si las características tienen diferentes escalas? ¡Deberíamos estandarizar!

Echemos un vistazo más de cerca a las columnas Age y Commission (in value). Imagina que para Age los valores posibles están en el rango de 0 a 100. Para Commission (in value), los valores son de $100 a $1000. La magnitud de los valores y la dispersión es mayor para la columna Commission (in value), lo que significa que el algoritmo encontrará que la característica Commission (in value) es más importante que la de Age. No queremos eso. Todas las características deben considerarse igualmente importantes antes de la ejecución del algoritmo.

Este problema se puede solucionar con el escalado de características.

Una de las formas de escalar las características es estandarizar los datos.

Piensa que todas las características se distribuyen normalmente, la media (M) y la varianza (Var) se determinan a partir de la muestra. Los valores de las características se convierten mediante esta fórmula:



Para la nueva característica, la media se convierte en 0 y la varianza es igual a 1.

Hay una clase *sklearn* dedicada para la estandarización de datos que se llama *StandardScaler*. Está en el módulo sklearn.preprocessing.

Importa StandardScaler de la librería:

from sklearn.preprocessing import StandardScaler

Crea una instancia de la clase y ajústala usando los datos de entrenamiento. El proceso de ajuste implica calcular la media y la varianza:

scaler = StandardScaler()

scaler.fit(features\_train)

Transforma el conjunto de entrenamiento y el conjunto de validación usando transform(). Almacena los conjuntos modificados en variables de la siguiente manera: features\_train\_scaled y features\_valid\_scaled:

features\_train\_scaled = scaler.transform(features\_train)

features\_valid\_scaled = scaler.transform(features\_valid)

Estandariza las características numéricas. Importa StandardScaler desde el módulo sklearn.preprocessing.

Crea una instancia de la clase StandardScaler() y ajústala usando los datos de entrenamiento. (La variable numeric ya contiene la lista de todas las características numéricas).

Almacena los conjuntos de entrenamiento y validación modificados en variables de la siguiente manera: features\_train y features\_valid.

Muestra en pantalla las primeras cinco filas.

Cuando estandarices las funciones, es posible que te encuentres con SettingWithCopyWarning. Esta no te impedirá pasar la tarea, pero si lo deseas, puedes silenciarla con la siguiente declaración: pd.options.mode.chained\_assignment = None

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

# < escribe tu código aquí >

data = pd.read\_csv('/datasets/travel\_insurance\_us.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

numeric = ['Duration', 'Net Sales', 'Commission (in value)', 'Age']

scaler = StandardScaler()

scaler.fit(features\_train[numeric])# < escribe tu código aquí >

features\_train[numeric] = scaler.transform(features\_train[numeric])

features\_valid[numeric] = scaler.transform(features\_valid[numeric])

# < transforma el conjunto de validación >

print(features\_train.head(5))

Resultado

Agency Agency Type ... Gender Age

33312 EPX Travel Agency ... None -0.283389

50154 EPX Travel Agency ... None -0.283389

26729 EPX Travel Agency ... None -0.709982

37842 EPX Travel Agency ... None -0.354488

23588 EPX Travel Agency ... None -0.425587

[5 rows x 10 columns]

¡Es correcto!

Edad = -0.28. ¿Es este el primer caso de viaje en el tiempo? ¿O tal vez es un seguro prenatal? Muy de ciencia ficción, ja, ja, ja. Hablando en serio, hemos estandarizado con éxito nuestras características.

Capítulo 2/8 · Faltan 2 lecciones

Codificación y estandarización de datos

# Cierre

### ¡Ahora eres profesional de la Data Science en un 80 %! ¡Es broma! Todavía no, ¡pero has recorrido un largo camino!

Combinemos la codificación de características categóricas y la estandarización de características numéricas y veamos el código final:

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

data = pd.read\_csv('travel\_insurance\_us.csv')

data\_ohe = pd.get\_dummies(data, drop\_first=True)

target = data\_ohe['Claim']

features = data\_ohe.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345)

numeric = ["Duration", "Net Sales", "Commission (en valor)", "Age"]

scaler = StandardScaler()

scaler.fit(features\_train[numeric])

features\_train[numeric] = scaler.transform(features\_train[numeric])

features\_valid[numeric] = scaler.transform(features\_valid[numeric])

print(features\_train.shape)

(37995, 188)

Pregunta

¿Qué está pasando en el siguiente código?

data\_ohe = pd.get\_dummies(data, drop\_first=True)

Escalado de datos

Carga de datos

Codificación One-Hot de características categóricas

¡Correcto! Es OHE.

División de datos en un conjunto de entrenamiento y un conjunto de validación

Codificación ordinal de características categóricas

División de datos en un conjunto de entrenamiento y un conjunto de prueba

¡Perfecto!

Pregunta

¿Qué está pasando en el siguiente código?

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(features, target, test\_size=0.25, random\_state=12345)

Escalado de datos

Carga de datos

Codificación ordinal de características categóricas

División de datos en un conjunto de entrenamiento y un conjunto de prueba

Codificación One-Hot de características categóricas

División de datos en un conjunto de entrenamiento y un conjunto de validación

¡Correcto! Necesitamos este paso para probar el modelo y ajustar los hiperparámetros.

¡Excelente trabajo!

Pregunta

¿Qué está pasando en el siguiente código?

scaler = StandardScaler()

scaler.fit(features\_train[numeric])

features\_train[numeric] = scaler.transform(features\_train[numeric])

features\_valid[numeric] = scaler.transform(features\_valid[numeric])

Carga de datos

Codificación One-Hot de características categóricas

Escalado de datos

¡Correcto! Estandarizamos las funciones para facilitar que los algoritmos entrenen modelos.

División de datos en un conjunto de entrenamiento y un conjunto de validación

Codificación ordinal de características categóricas

División de datos en un conjunto de entrenamiento y un conjunto de prueba

¡Perfecto!

# Conclusión

En este capítulo aprendiste a hacer lo siguiente:

* Codificar características categóricas con *OHE.*
* Transformar características categóricas con *codificación ordinal.*
* Escalar características numéricas.

En el próximo capítulo descubrirás por qué la métrica *exactitud* no es adecuada para resolver nuestro problema de seguros y aprenderás sobre las otras métricas que tenemos a nuestra disposición.

### Llévate esto contigo

Descárgate el resumen del capítulo y la hoja informativa para poder consultarlos cuando los necesites.

* [Resumen del capítulo: Preparación de características](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_8/esp/moved_Hoja_informativa_preparacin_de_caractersticas_Takeaway_sheet_Feature_Preparation_esp.pdf)
* [Hoja informativa: Preparación de características](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_8/esp/moved_Resumen_del_captulo_preparacin_de_caractersticas.pdf)

Capítulo 3/8

Métricas de clasificación

# Introducción

### Los datos están listos, así que es hora de elegir una métrica.

## Empezarás por:

* Averiguar cuándo no funciona la métrica de *exactitud* (accuracy).
* Aprender a trazar una matriz de confusión.
* Trabajar con nuevas métricas: precisión, recall y valor *F1*.

## ¿Cuánto tiempo tomará?

14 lecciones de 5-10 minutos cada una

## Descripción del ejercicio

Sigamos con la tarea de seguros en la que hemos estado trabajando.

Las siguientes lecciones contendrán mucha más práctica que teoría. ¿Todo listo para ensuciarte las manos?

# Exactitud para el árbol de decisión

### Primero, debemos verificar si la métrica de exactitud es adecuada para la tarea. Entrenemos al modelo y veamos.

Podemos calcular la exactitud del modelo con la función accuracy\_score() . Esta toma respuestas y predicciones correctas y devuelve la proporción de respuestas clasificadas correctamente.

Hemos guardado el conjunto de datos con las funciones preparadas en el archivo travel\_insurance\_us\_preprocessed.csv.

Entrena el modelo de árbol de decisión calculando el valor de exactitud en el conjunto de validación. Para hacerlo, declara la variable predicted\_valid, luego guarda el resultado en la variable accuracy\_valid.Especifica random\_state=12345. Muéstrala en la pantalla.

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = DecisionTreeClassifier(random\_state=12345)

model.fit(features\_train, target\_train)

predicted\_valid = model.predict(features\_valid)

# Calcular la exactitud

accuracy\_valid = accuracy\_score(target\_valid, predicted\_valid)

# Mostrar la exactitud en pantalla

print(accuracy\_valid)  
  
Resultado

0.9745756020529017

¡Es correcto!

¡Esa es una exactitud de primera clase! El modelo está entrenado, pero ¿estará listo para entrar a la adultez?

# Prueba de consistencia

### La proporción de respuestas correctas es del 97%. ¿Es suficiente? Comparemos con la variable objetivo.

Para evaluar la cordura del modelo, verifiquemos con qué frecuencia la variable objetivo contiene la clase "1" o "0". El número de valores únicos se calcula utilizando el método value\_counts(), que agrupa exactamente los mismos valores.

1.

Para contar clases en la característica objetivo, utiliza el método value\_counts(). Haz las frecuencias relativas (de 0 a 1). La documentación de Pandas te ayudará a hacerlo.

Guarda los valores en la variable class\_frequency. Muéstralos en pantalla

Usa el método plot() con el argumento kind='bar'' para trazar un diagrama.

import pandas as pd

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

#Calcular las frecuencias relativas de las clases en la variable objetivo

class\_frequency = data['Claim'].value\_counts(normalize=True)

# Mostrar las frecuencias relativas en pantalla

print(class\_frequency)

# Trazar un diagrama de barras de las frecuencias relativas

class\_frequency.plot(kind='bar', title='Frecuencia de Clases en la Variable Objetivo')

Resultado

0 0.985136

1 0.014864

Name: Claim, dtype: float64

Gráfico, Gráfico de barras

Descripción generada automáticamente

¡Es correcto!

¡Que diferencia! Pero las estadísticas no mienten: solo el 1% de los clientes presentan una queja.

2.

Analiza las frecuencias de clase de las predicciones del árbol de decisión (la variable predicted\_valid).

De manera similar:

* Aplica el método value\_counts(). Relativiza las frecuencias.
* Guarda los valores en la variable class\_frequency. Muéstralos en pantalla.
* Usa el método plot() con el argumento kind='bar' para trazar un diagrama.

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = DecisionTreeClassifier(random\_state=12345)

model.fit(features\_train, target\_train)

# Predecir en el conjunto de validación

predicted\_valid = model.predict(features\_valid)

# Calcular la precisión

accuracy\_valid = accuracy\_score(target\_valid, predicted\_valid)

# Calcular las frecuencias relativas de las predicciones

class\_frequency = pd.Series(predicted\_valid).value\_counts(normalize=True)

print(class\_frequency)

# Trazar un diagrama de barras de las frecuencias relativas

class\_frequency.plot(kind='bar', title='Frecuencia de Clases en las Predicciones')  
  
Resultado

0 0.986024

1 0.013976

dtype: float64

¡Es correcto!

Nuestro modelo de árbol de decisión produce proporciones de clases que son similares a las que se observan simplemente eligiendo la clase más común en los datos, lo cual se aprecia en la primera parte del ejercicio donde analizamos la frecuencia de las clases en la variable objetivo. Para verificar si el modelo está realmente aprendiendo patrones útiles, es crucial compararlo con un modelo constante en una prueba de cordura.

Gráfico, Gráfico de barras

Descripción generada automáticamente

3.

Crea un modelo constante: predice la clase "0" para cualquier observación. Guarda sus predicciones en la variable target\_pred\_constant. Muestra en pantalla el valor de accuracy.

import pandas as pd

from sklearn.metrics import accuracy\_score

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

# Crear un modelo constante que predice la clase "0"

target\_pred\_constant = pd.Series([0] \* len(target))

# Calcular la precisión del modelo constante

accuracy\_constant = accuracy\_score(target, target\_pred\_constant)

# Mostrar la precisión en pantalla

print(accuracy\_constant)  
Resultado

0.9851362021318595

Es correcto!

"¡Qué gran modelo! ¡Podemos enviarlo a producción!" Esa suele ser la primera reacción, hasta que descubres que no pasa la prueba de cordura porque sus predicciones no son mejores que las de un modelo constante que siempre elige la clase más frecuente. Este resultado indica que el modelo no está capturando patrones significativos en los datos, lo cual es crucial para un desempeño fiable. ¿Eso significa que es hora de rendirse?

Sprint 9

Capítulo 3/8

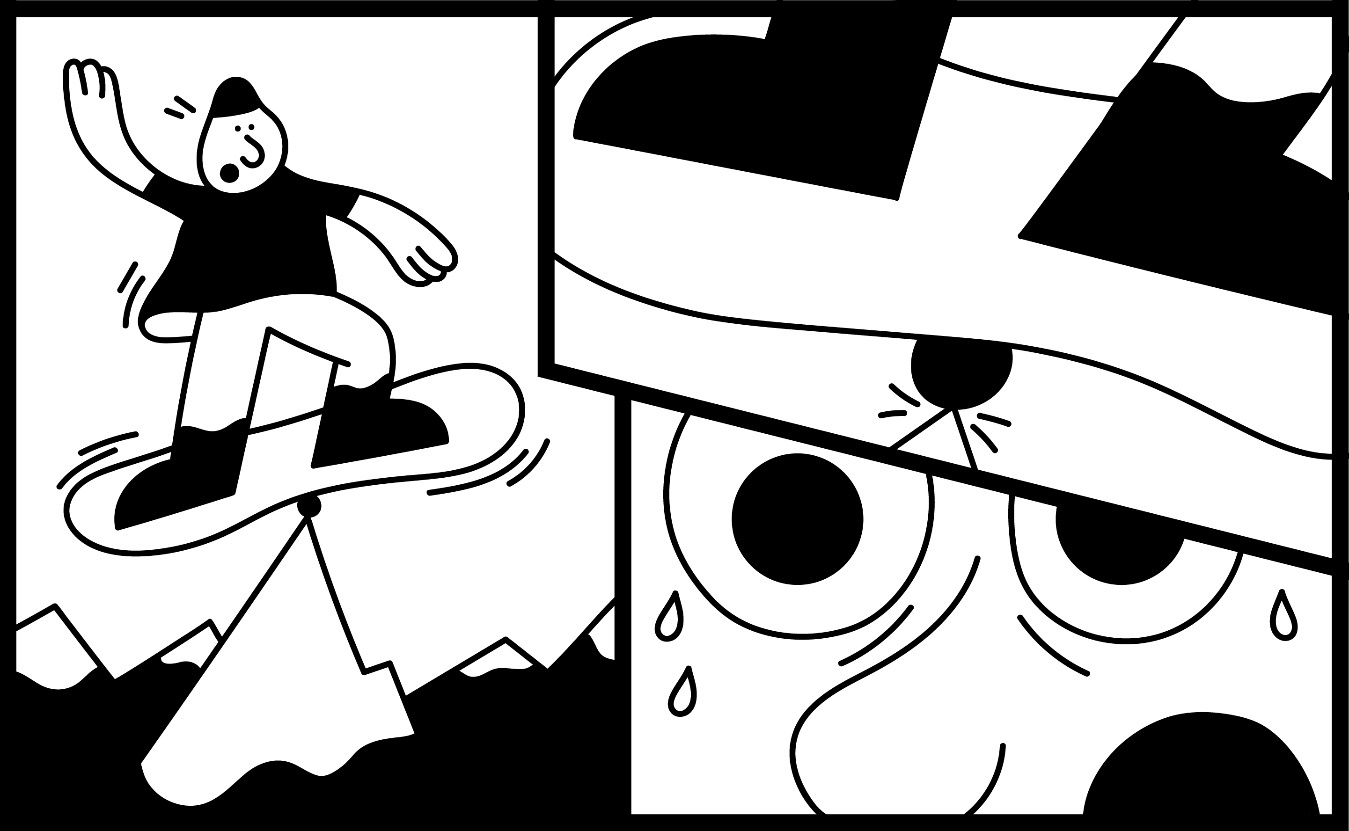
Métricas de clasificación

# Equilibrio y desequilibrio de las clases

### La *exactitud* del árbol de decisión es casi la misma que la del modelo constante. ¿Por qué pasa eso y qué debemos hacer al respecto?

Aunque la proporción de predicciones correctas es cercana al 100 %, todavía no sabemos si el cliente hará una reclamación al seguro. Hay un fuerte desequilibrio de clases en nuestro problema, lo que afecta cómo se entrena el modelo.

Las clases están desequilibradas cuando su proporción está lejos de 1:1. El equilibrio de clases se observa si su número es aproximadamente igual.



La exactitud no evalúa bien el modelo en un escenario de fuerte desequilibrio. ¡Necesitamos una nueva métrica! Pero primero debemos repasar algunas distinciones importantes.

Una clase con la etiqueta "1" se llama positiva y una clase con la etiqueta "0" se llama negativa.

Si combinamos estas respuestas con predicciones, obtendremos la siguiente división:

* Respuestas de verdadero positivo (VP) y verdadero negativo (VN)
* Respuestas de falso positivo (FP) y falso negativo (FN)

Entonces, las características "positivo" y "negativo" se refieren a una predicción, y "verdadero" y "falso" se refieren a su exactitud.

En las próximas lecciones, veremos cómo podemos calcular estos cuatro valores posibles.

# Respuestas verdaderas positivas

### Podemos utilizar diferentes métricas para gestionar el desequilibrio y clasificar las respuestas con mayor exactitud.

¿Qué significa una respuesta verdadera positiva (VP)? En este caso, el modelo etiquetó una observación como 1 y su valor real también es 1.

En nuestro ejercicio, la respuesta Verdadero Positivo equivale al número de asegurados que:

* pidió compensación, de acuerdo con la predicción del modelo;
* sí hizo un reclamo de seguro.

Aquí hay un ejemplo de predicciones frente a respuestas correctas. Cuenta el número de respuestas VP y muestra el resultado en la pantalla.

import pandas as pd

target = pd.Series([1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1])

predictions = pd.Series([1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1])

# Contar el número de Verdaderos Positivos (VP)

vp = ((target == 1) & (predictions == 1)).sum()

# Mostrar el resultado en pantalla

print(vp)

Resultado

5

¡Es correcto!

¡Conociste a Verdadero Positivo, el líder del cuarteto! Intenta no confundirlo. Faltan otros tres.

# Respuestas verdaderas negativas

### Si los valores de clase previstos y reales son negativos, la respuesta es Verdadero Negativo.

En nuestro ejercicio, la respuesta Verdadero Negativo *(VN)* es el número de personas aseguradas que:

* de acuerdo con la predicción del modelo, no solicitaron un pago
* en realidad no solicitaron la compensación del seguro

Cuenta el número de respuestas VN tal como lo hiciste en el ejercicio anterior. Muestra los resultados en la pantalla.

import pandas as pd

target = pd.Series([1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1])

predictions = pd.Series([1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1])

print(((target == 0) & (predictions == 0)).sum())

# < escribe el código aquí >

Resultado

4

¡Es correcto!

¡Es correcto!

Imagina cambiar a los miembros del cuarteto. Intenta no confundirlos por los Cuatro Jinetes del Apocalipsis. Este definitivamente montaría un caballo blanco. Y su jinete, Verdadero Negativo.

# Respuestas falsas positivas

### Los algoritmos son como los humanos: cometen errores. Estos errores se dividen en dos categorías.

El primer tipo de error es un falso positivo (*FP*). Ocurre cuando el modelo predijo "1", pero el valor real de la clase es "0".

En nuestra tarea, una respuesta Falso Positivo es el número de personas aseguradas que:

* según la predicción del modelo, solicitaron un pago, pero
* en realidad, no hicieron un reclamo.

# Respuestas falsas positivas

### Los algoritmos son como los humanos: cometen errores. Estos errores se dividen en dos categorías.

El primer tipo de error es un falso positivo (*FP*). Ocurre cuando el modelo predijo "1", pero el valor real de la clase es "0".

En nuestra tarea, una respuesta Falso Positivo es el número de personas aseguradas que:

* según la predicción del modelo, solicitaron un pago, pero
* en realidad, no hicieron un reclamo.

Encuentra el número de respuestas *FP* de la misma manera que encontraste las respuestas *VN* en la tarea anterior. Muestra los resultados en la pantalla.  
  
Sustituye los signos de interrogación por los valores apropiados:

print(((target == ???) & (predictions == ???)).sum())

Si la respuesta es un falso positivo, entonces la predicción es "1".

import pandas as pd

target = pd.Series([1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1])

predictions = pd.Series([1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1])

print(((target == 0) & (predictions == 1)).sum())

# < escribe el código aquí >  
Resultado

3

¡Es correcto!

Similar a un falso profeta, el Falso Positivo es... Espera, esto no es la escuela dominical. ¡Muy bien! Al igual que el Verdadero Positivo, el Falso Positivo es positivo, pero no es verdadero.

# Respuestas falsas negativas

### El segundo tipo de error son las respuestas Falsas Negativas (FN).

Las respuestas falsas negativas ocurren cuando el modelo predice "0", pero el valor real de la clase es "1".

En nuestra tarea, una respuesta de Falso Negativo es el número de personas aseguradas que:

* según la predicción del modelo, no solicitaron un pago, pero
* en realidad, hicieron un reclamo.

Cuenta el número de respuestas *FN*. Muestra los resultados en la pantalla.  
  
import pandas as pd

target = pd.Series([1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1])

predictions = pd.Series([1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1])

print(((target == 1) & (predictions == 0)).sum())

Resultado

2

¡Es correcto!

¡Es correcto!

Tanto Falso como Negativo suenan negativos, pero juntos significan positivo. Y es maravilloso. Camina sobre las huellas de lo Falso y lo Negativo y encuentra el bien en los dos.

# Matriz de confusión

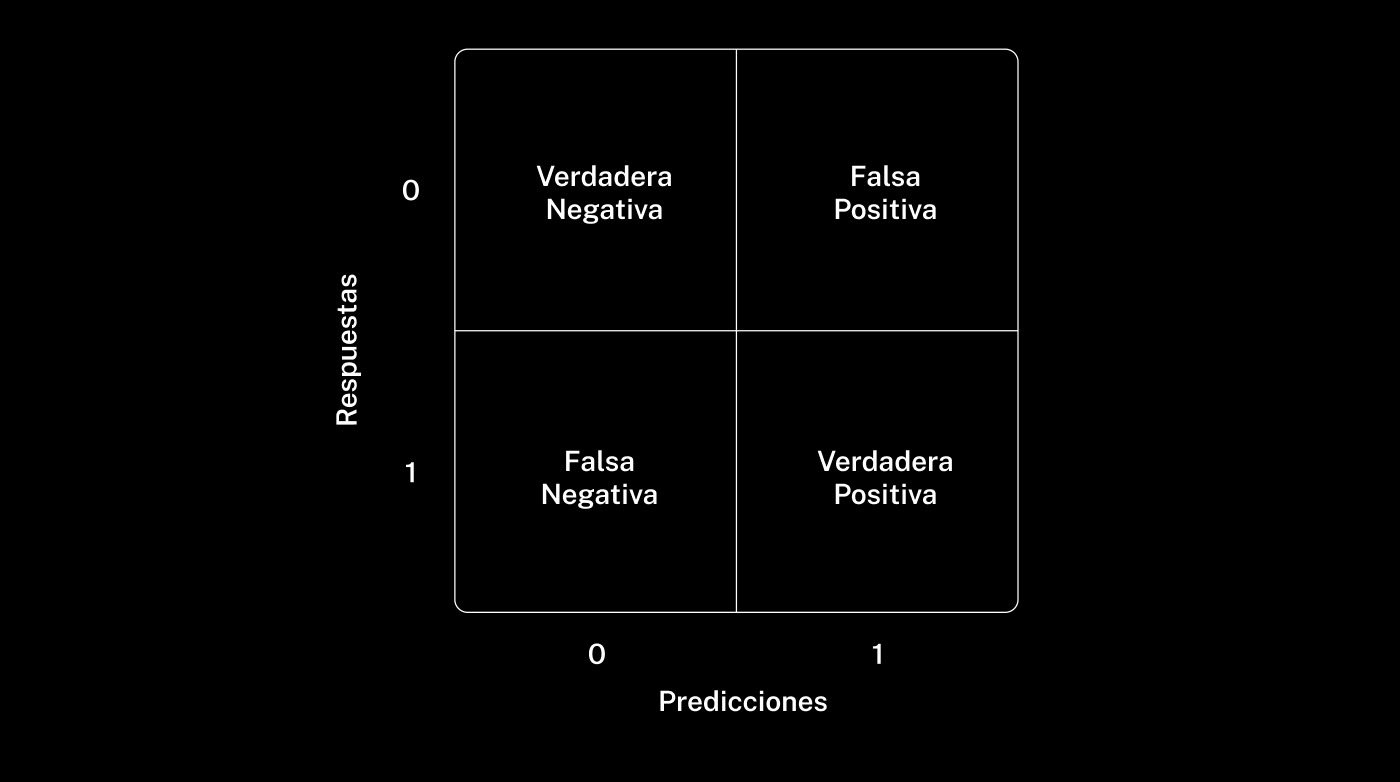
### Cuando *VP, FP, VN, FN* se recopilan en una tabla, se denomina *matriz de confusión*.

La matriz se forma de la siguiente manera:

* las etiquetas del algoritmo (0 y 1) se colocan en el eje horizontal ("Predicciones");
* las etiquetas verdaderas de la clase (0 y 1) se colocan en el eje vertical ("Respuestas").

Lo que obtienes:

1. Las predicciones correctas están en la diagonal principal (desde la esquina superior izquierda):
   * *VN* en la esquina superior izquierda
   * *VP* en la esquina inferior derecha
2. Las predicciones incorrectas están fuera de la diagonal principal:
   * *FP* en la esquina superior derecha
   * *FN* en la esquina inferior izquierda



La matriz de confusión te permite visualizar los resultados de calcular las métricas de *exactitud* y *recall*.

La matriz de confusión se encuentra en el módulo sklearn.metrics, que ya conoces. La función confusion\_matrix() toma respuestas y predicciones correctas y devuelve una matriz de confusión.

.

Calcula la matriz de confusión utilizando la función *confusion\_matrix()*. Impórtala desde el módulo *sklearn.metrics*. Muestra los resultados en la pantalla.

import pandas as pd

from sklearn.metrics import confusion\_matrix# < escribe el código aquí >

target = pd.Series([1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1])

predictions = pd.Series([1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1])

matriz = confusion\_matrix(target, predictions)

print(matriz)# < escribe el código aquí >

Resultado

[[4 3]

[2 5]]

¡Es correcto!

¡La matriz de confusión es el hogar perfecto para Verdadero Positivo, Falso Negativo, Falso Positivo y Verdadero Negativo!

2.

Calcula una matriz de confusión para el árbol de decisión y llama a la función confusion\_matrix(). Muéstrala en pantalla. Muéstrala en pantalla.

Utiliza la misma plantilla de código que en la tarea anterior. Reemplaza target y predictions con target\_valid y predicted\_valid.  
  
import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = DecisionTreeClassifier(random\_state=12345)

model.fit(features\_train, target\_train)

predicted\_valid = model.predict(features\_valid)

matriz = confusion\_matrix(target\_valid, predicted\_valid)

print(matriz)# < escribe el código aquí >

# < escribe el código aquí >  
Resultado

[[12331 165]

[ 157 12]]

Es correcto!

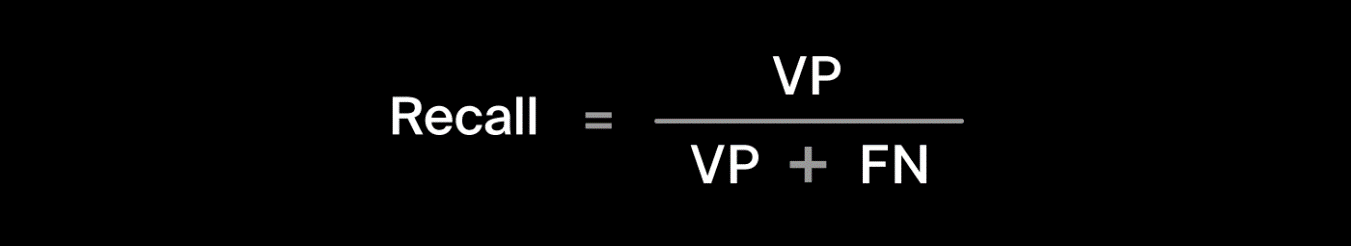
El modelo es bastante pesimista. A menudo ve respuestas negativas donde no debería.¿Esto te recuerda a alguien?

# Recall

### La matriz de confusión te ayudará a crear nuevas métricas. Comencemos con **recall**.

*Recall* revela la porción de respuestas positivas identificadas por el modelo o la proporción de respuestas positivas marcadas como positivas por el modelo (VP) frente a las respuestas positivas marcadas como positivas por el modelo (VP) más las respuestas marcadas como negativas por el modelo que en realidad son positivas (FN). Estas respuestas positivas son valiosas, por lo que es importante saber con qué eficacia las encuentra el modelo.

*Recall* se calcula usando esta fórmula:



Veamos un ejemplo:

* 100 personas aseguradas hicieron reclamaciones. Este es el número de todas las observaciones positivas o *VP* *+* *FN,*
* El modelo identificó correctamente solo 20
* entonces, *recall* es 0.2.

*Recall* es la proporción de VP entre todas las respuestas que tienen una etiqueta verdadera de 1. Queremos que el valor de recall esté cerca de 1. Esto significaría que el modelo es bueno para identificar verdaderos positivos. Si está más cerca de cero, el modelo necesita ser revisado y arreglado.

se encarga de calcular recall. Impórtala.

La función toma respuestas y predicciones correctas y devuelve la proporción de respuestas correctas encontradas por el modelo. Muestra los resultados en la pantalla.

Es la función recall\_score().

from sklearn.metrics import recall\_score

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import recall\_score

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345)

model = DecisionTreeClassifier(random\_state=12345)

model.fit(features\_train, target\_train)

predicted\_valid = model.predict(features\_valid)

matriz = recall\_score(target\_valid, predicted\_valid)

print(matriz)# < escribe el código aquí >

# < escribe el código aquí >

2.

Pasa por los valores de umbral de 0 a 0.3 en intervalos de 0.02. Encuentra precisión y recall para cada valor del umbral. Muestra en pantalla los resultados (en precódigo)

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

Más adelante aprenderemos más sobre las herramientas de la librería *numpy*.

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import precision\_score, recall\_score

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = LogisticRegression(random\_state=12345, solver='liblinear')

model.fit(features\_train, target\_train)

probabilities\_valid = model.predict\_proba(features\_valid)

probabilities\_one\_valid = probabilities\_valid[:, 1]

for threshold in np.arange(0, 0.3, 0.02):

predicted\_valid = probabilities\_one\_valid > threshold

precision = precision\_score(target\_valid, predicted\_valid)

recall = recall\_score(target\_valid, predicted\_valid)

print('Threshold = {:.2f} | Precision = {:.3f}, Recall = {:.3f}'.format(threshold, precision, recall))  
  
Resultado

Threshold = 0.00 | Precision = 0.013, Recall = 1.000

Threshold = 0.02 | Precision = 0.052, Recall = 0.645

Threshold = 0.04 | Precision = 0.061, Recall = 0.609

Threshold = 0.06 | Precision = 0.072, Recall = 0.367

Threshold = 0.08 | Precision = 0.097, Recall = 0.254

Threshold = 0.10 | Precision = 0.112, Recall = 0.178

Threshold = 0.12 | Precision = 0.146, Recall = 0.107

Threshold = 0.14 | Precision = 0.033, Recall = 0.012

Threshold = 0.16 | Precision = 0.036, Recall = 0.006

Threshold = 0.18 | Precision = 0.000, Recall = 0.000

Threshold = 0.20 | Precision = 0.000, Recall = 0.000

Threshold = 0.22 | Precision = 0.000, Recall = 0.000

Threshold = 0.24 | Precision = 0.000, Recall = 0.000

Threshold = 0.26 | Precision = 0.000, Recall = 0.000

Threshold = 0.28 | Precision = 0.000, Recall = 0.000

¡Es correcto!

* El recall para el umbral 0.0 es uno: todas las respuestas son positivas. A medida que el umbral aumenta, el recall disminuye porque se predicen menos respuestas como positivas.
* La precisión inicialmente aumenta a medida que el umbral se eleva, debido a una selección más estricta de las respuestas positivas. Sin embargo, después cae a cero cuando ya no se predice ninguna respuesta como clase "1".

¿Deberíamos elevar más el umbral? ¡Inútil! El modelo ha dejado de predecir la clase positiva y los valores de ambas métricas son cero.

Sprint 9

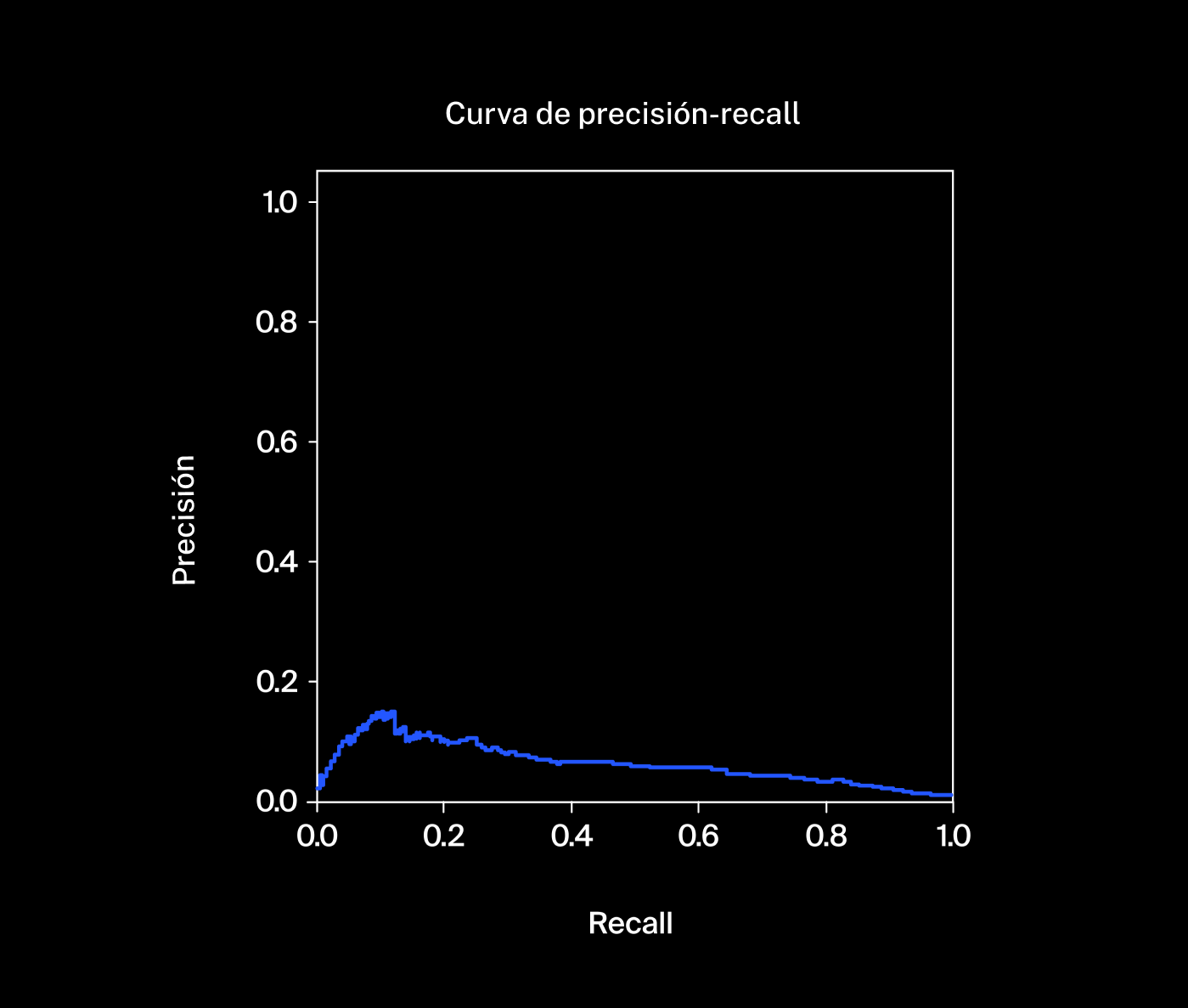
Capítulo 4/8

Clasificación desbalanceada

# 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.



¿Quieres saber cómo se obtiene la curva? Aquí tienes el código:

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import precision\_recall\_curve

from sklearn.linear\_model import LogisticRegression

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = LogisticRegression(random\_state=12345, solver='liblinear')

model.fit(features\_train, target\_train)

probabilities\_valid = model.predict\_proba(features\_valid)

precision, recall, thresholds = precision\_recall\_curve(

target\_valid, probabilities\_valid[:, 1]

)

plt.figure(figsize=(6, 6))

plt.step(recall, precision, where='post')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.ylim([0.0, 1.05])

plt.xlim([0.0, 1.0])

plt.title('Precision-Recall Curve')

plt.show()

En la lección 9 analizaremos este código en detalle.

Resultado

0.07100591715976332

¡Es correcto!

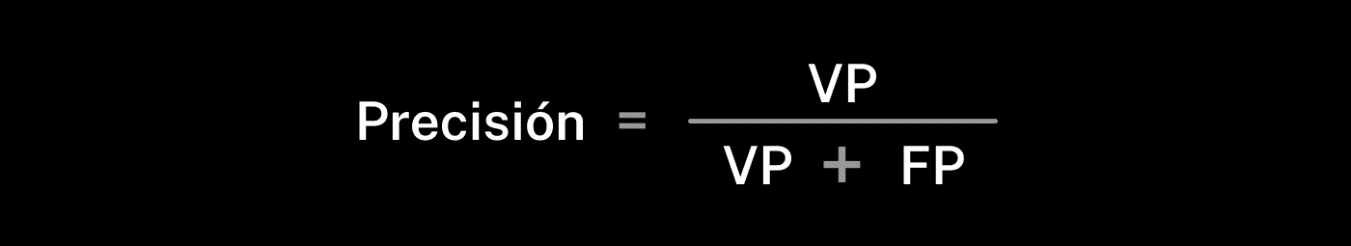
¡Qué fracaso! El valor de recall no se acerca a 1. ¡Tal vez la métrica de precisión pueda ayudar!

# recisión

### Otra métrica para evaluar la calidad de una predicción de clase objetivo es la **precisión**.

La precisión mide la relación entre las predicciones verdaderas positivas y todas las predicciones positivas realizadas por el modelo. Por lo tanto, cuantas más predicciones de falsos positivos se hagan, menor será la *precisión*.

La *precisión* se calcula usando esta fórmula:



Veamos un ejemplo:

* Según la predicción del modelo, 100 personas aseguradas solicitarán una indemnización. Este es el número de todas las observaciones que el modelo ha etiquetado como positivas, o *VP + FP*
* 20 de ellas realmente solicitaron un pago de seguro (este es el número de VP);
* la *precisión* es 0.2.

Recuerda que *VP* representa respuestas verdaderas positivas. FP representa respuestas positivas marcadas por el modelo. Necesitamos que la *precisión* esté cerca de uno.

En el módulo *sklearn.metrics*, encuentra la función que calcula la *precisión*. Impórtala.

Esta función toma respuestas y predicciones correctas. Devuelve observaciones marcadas como positivas por el modelo que en realidad son positivas. Muestra los resultados en la pantalla.

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import precision\_score# < write code here >

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = DecisionTreeClassifier(random\_state=12345)

model.fit(features\_train, target\_train)

predicted\_valid = model.predict(features\_valid)

matriz = precision\_score(target\_valid, predicted\_valid)

print(matriz)

# < escribe el código aquí >  
  
Resultado

0.06779661016949153

¡Es correcto!

La métrica de *precisión* tampoco nos dio lo que queríamos. ¡Nos estamos quedando sin opciones!

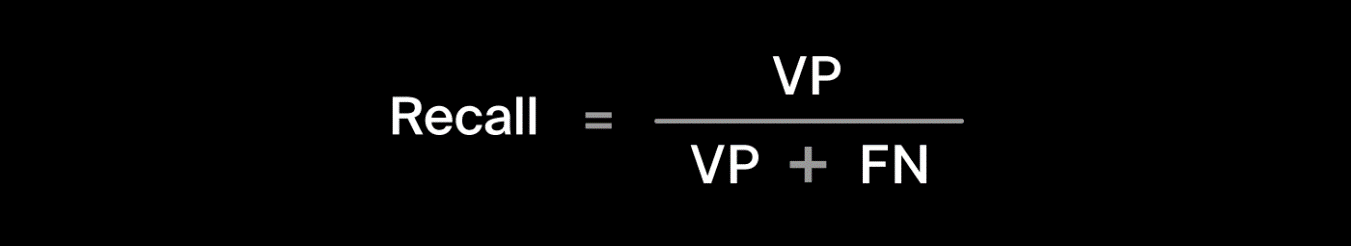
Capítulo 3/8 · Faltan 3 lecciones

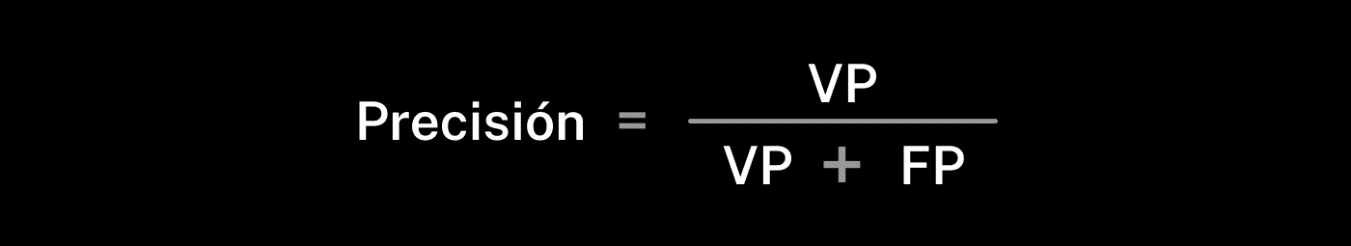
Métricas de clasificación

# Recall vs. Precisión

### Cuando el modelo predice mal las clases positivas, tanto la recall como la precisión son bajas. ¿Podemos mejorar sus valores?

Veamos de nuevo las fórmulas para calcular *recall* y *precisión*:





Pregunta

Si *recall* es la métrica clave en la pregunta anterior, ¿cómo puedes hacer que sea lo más alta posible?

Entrena un modelo que prediga "0" para todas las observaciones. *Recall* será igual a 0.5.

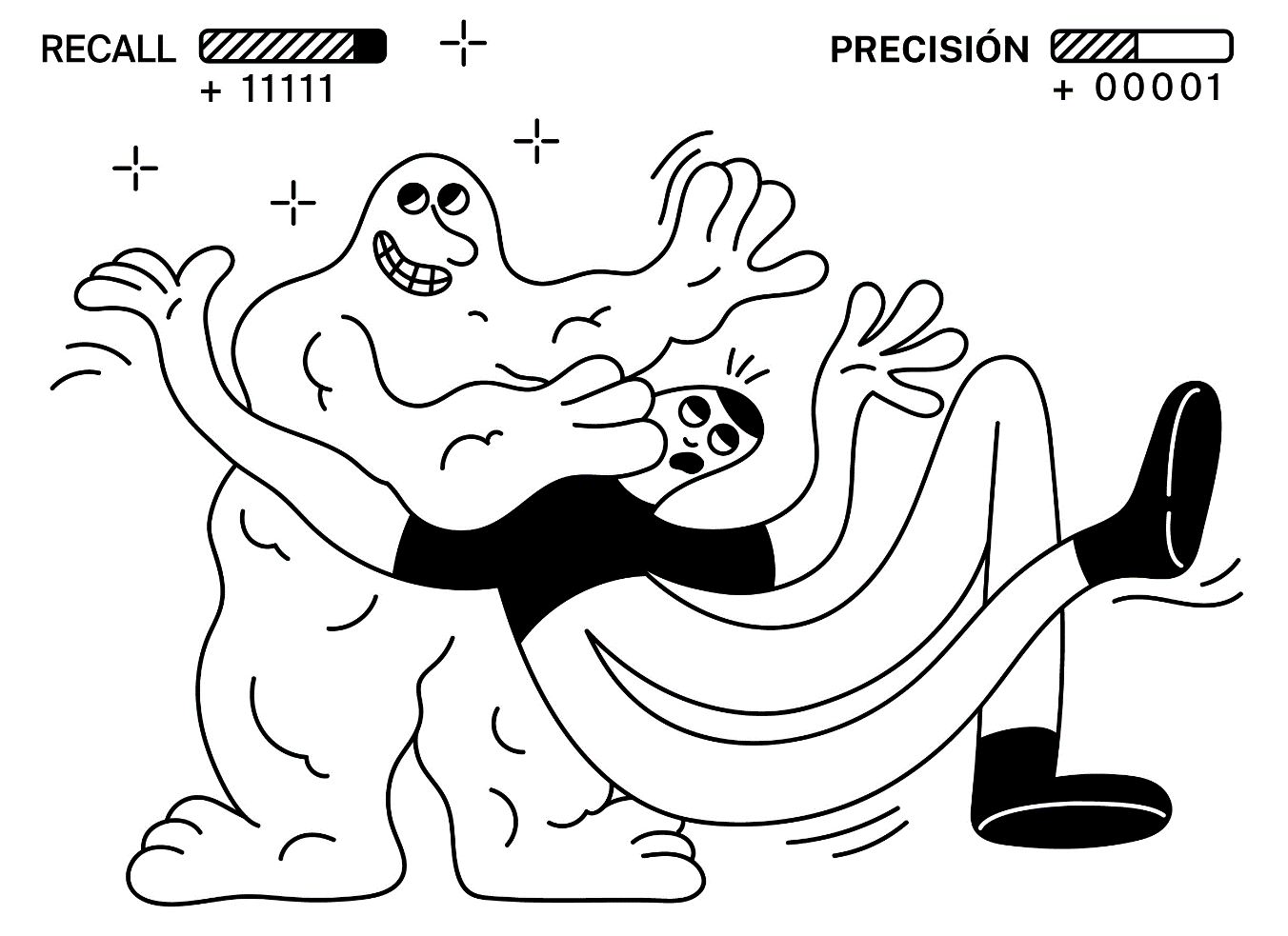
Entrena un modelo que prediga "0" para todas las observaciones. *Recall* será igual a 1.0

Entrena un modelo que prediga "1" para todas las observaciones. *Recall* será igual a 0.5

Entrena un modelo que prediga "1" para todas las observaciones. Recall será igual a 1.0

¡Correcto! Si el modelo predice "1" para todas las observaciones, el valor de FN será 0, lo que dejará solo VP en el denominador de la fórmula, por lo que recall será 1 (VP/VP).

¡Buen trabajo!



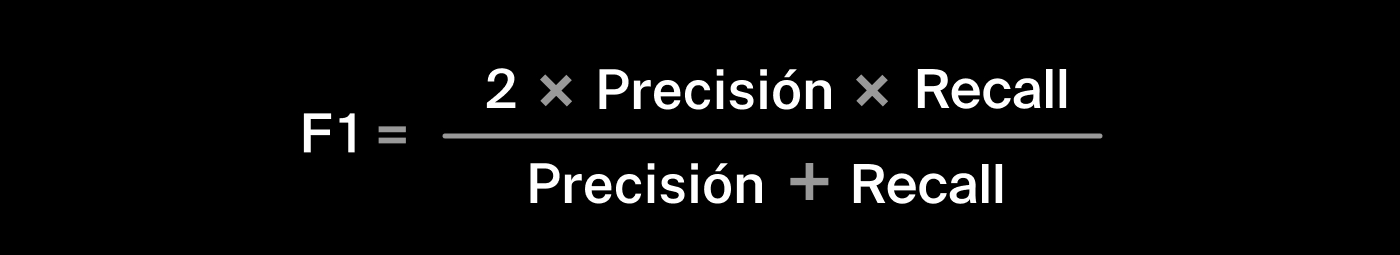
¡Ya descubrimos cómo maximizar la métrica de *recall*! La fórmula tiene en cuenta solo el error de la clase positiva, pero ignora la clase negativa. Debes entrenar un modelo que responda "1" con la menor frecuencia posible. Pero responder siempre "0" no ayudará, porque entonces dividirás cero entre cero en la fórmula.

# Valor F1

### Algunas veces, por separado, recall y precisión no son muy informativas. Debes aumentar simultáneamente los valores de ambas métricas... ¡O podemos recurrir a una nueva métrica que las combine!

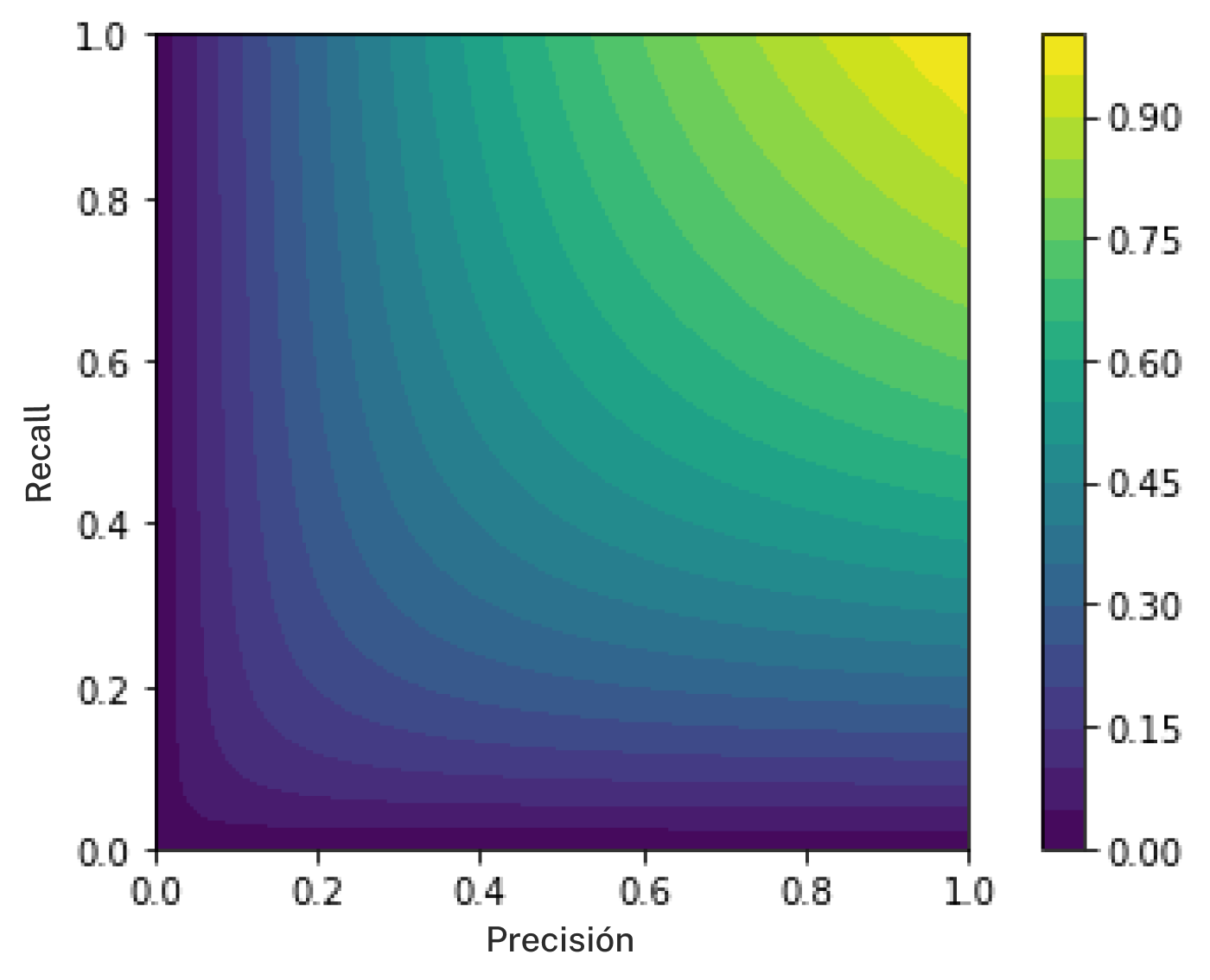
*Recall* y *precision* evalúan la calidad de las predicciones de la clase positiva desde diferentes ángulos. *Recall* describe qué tan bien comprende un modelo las propiedades de esta clase y es capaz de reconocerla. *Precisión* detecta si un modelo está exagerando el reconocimiento de clase positiva al asignar demasiadas etiquetas positivas.

Ambas métricas son importantes. Las métricas de agregación, una de los cuales es el valor F1, ayudan a controlarlas simultáneamente. Esta es la media armónica de *recall* y *precisión*. En *F1,* 1 significa que la relación de *recall* a *precisión* es 1:1.



Es importante comprender que cuando *recall* o *precisión* están cerca de cero, la media armónica misma se aproxima a 0.

La siguiente gráfica muestra los valores del valor *F1* para diferentes valores de *recall* y *precisión*. El color azul corresponde a cero y el color amarillo corresponde a uno.



Si una clase positiva es pobremente predicha en una de las escalas (recall o precisión), entonces un valor F1 cercano a cero mostrará que la predicción de la clase 1 ha fallado.

1.

Calcula lo siguiente:

* *precisión*, usando la función *precision\_score()*
* *recall*, usando la función *recall\_score()*
* Valor *F1*, utilizando la fórmula de la lección.

Asigna los valores de las métricas a las variables precision*,* recall y f1*.*

Finalmente, muéstralos en pantalla (ya en el precódigo).

2.

En el módulo *sklearn.metrics*, busca la función que calcula el valor *F1*. Impórtala.

Esta función toma las respuestas y predicciones correctas y devuelve la media armónica de *recall* y *precisión*. Muestra los resultados en la pantalla.

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import precision\_score# < write code here >

from sklearn.metrics import recall\_score

from sklearn.model\_selection import train\_test\_split

# < importa las funciones de sklearn.metrics >

target = pd.Series([1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1])

predictions = pd.Series([1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1])

precision = precision\_score(target, predictions)# <escribe el código aquí >

recall = recall\_score(target, predictions)# < escribe el código aquí >

f1 = 2 \* precision \* recall / (precision + recall)# <escribe el código aquí >

print('Recall:', recall)

print('Precisión:', precision)

print('Puntuación F1', f1)

Resultado

Recall: 0.7142857142857143

Precisión: 0.625

Puntuación F1 0.6666666666666666

Es correcto!

*F1 es el promedio*. Quiero decir que no tenemos ni *recall* ni *precisión*, sino algo intermedio.

2.

En el módulo *sklearn.metrics*, busca la función que calcula el valor *F1*. Impórtala.

Esta función toma las respuestas y predicciones correctas y devuelve la media armónica de *recall* y *precisión*. Muestra los resultados en la pantalla.

Utiliza esta función en *sklearn.metrics* para calcular el valor *F1*:

from sklearn.metrics import f1\_score

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import f1\_score# < escribe el código aquí >

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = DecisionTreeClassifier(random\_state=12345)

model.fit(features\_train, target\_train)

predicted\_valid = model.predict(features\_valid)

f1 = f1\_score(target\_valid, predicted\_valid)

print(f1)

# < escribe el código aquí >

Resultado

0.06936416184971098

¡Es correcto!

Parece que el valor *F1* no pasó la prueba hoy. ¡La próxima vez lo haremos mejor!

Sprint 9

Capítulo 3/8 · Última lección

Métricas de clasificación

# Conclusión

### Has aprendido a medir la calidad de diferentes métricas y notar errores en las tareas.

### En este capítulo, también aprendiste:

* qué es el desequilibrio de clases;
* cómo trazar una matriz de confusión de predicciones;
* cuándo usar *precisión*, *recall* y valor *F1*.

En el próximo capítulo aprenderás cómo superar el problema del desequilibrio de clases y ajustar tu modelo para maximizar el valor *F1*.

### Llévate esto contigo

Descárgate el resumen del capítulo y la hoja informativa para poder consultarlos cuando los necesites.

* [Resumen del capítulo: Métricas de clasificación](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_8/Resumen_del_captulo_metricas_de_clasificacion.pdf?etag=650af8923907aa9679de799aff32960b)
* [Hoja informativa: Métricas de clasificación](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_8/moved_Hoja_informativa_mtricas_clasificacin.pdf)

Sprint 9

Capítulo 4/8

Clasificación desbalanceada

# Introducción

### En el capítulo anterior obtuvimos valores de nuestras métricas cercanos a cero. ¡Ayudemos a nuestro modelo!

## Empezarás por:

* Dominar las técnicas de ajuste de peso de clase (*sobremuestreo* y *submuestreo*) para mejorar la calidad del modelo en un escenario de desequilibrio de clases.
* Ajustar las métricas de clasificación considerando las probabilidades de clase.
* Trazar la curva *ROC* y calcular el área debajo de esta.

## ¿Cuánto tiempo tomará?

Diez lecciones de 10 minutos cada una.

## Descripción del ejercicio

Seguiremos trabajando en la tarea de seguros.

# Ajuste de peso de clase

### ¡Hagamos que las clases raras pesen más!

Imagina que estás haciendo un examen. Dependiendo de la dificultad, las respuestas correctas pueden darte uno o dos puntos. Para obtener una puntuación más alta, te enfocas solo en las respuestas de dos puntos. De manera similar, los modelos también buscan una puntuación más alta, por lo que tienden a centrarse en las observaciones de mayor importancia.

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.

El algoritmo de regresión logística en la librería sklearn tiene el argumento class\_weight. Por defecto, es None, es decir, las clases son equivalentes:

class "0" weight = 1.0

class "1" weight = 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:

class "0" weight = 1.0

class "1" weight = N

La clase rara tendrá un mayor peso.

Los árboles de decisión y los bosques aleatorios también tienen el argumento class\_weight.

Ejercicio

Aquí hay un código de entrenamiento para un modelo de regresión logística con clases igualmente ponderadas de las lecciones anteriores. Equilibra los pesos de clase pasando el valor de argumento adecuado para *class\_weight*. Observa cómo cambia el valor *F1*.

import pandas as pd

from sklearn.metrics import f1\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = LogisticRegression(random\_state=12345, class\_weight='balanced', solver='liblinear')

model.fit(features\_train, target\_train)

predicted\_valid = model.predict(features\_valid)

print('F1:', f1\_score(target\_valid, predicted\_valid))

Resultado

F1: 0.08698830409356724

¡Es correcto!

El valor *F1* ahora es 0.09. No es genial, pero tampoco terrible.

# Sobremuestreo

### ¿Cómo podemos hacer que las observaciones de una clase rara sean menos raras en los datos?

Ahora obtienes 1 punto por resolver cualquier tarea en la prueba. Las tareas más importantes se repiten varias veces para que sean más fáciles de recordar.

En el entrenamiento de modelos, esta técnica se denomina sobremuestreo.

El sobremuestreo se realiza en varios pasos:

* Dividir el conjunto de datos de entrenamiento en observaciones negativas y positivas;
* Duplicar las observaciones positivas (las que tienen ocurrencias raras) varias veces;
* Crear una nueva muestra de entrenamiento basada en los datos obtenidos;
* Barajar los datos: preguntas idénticas que se suceden no ayudarán al entrenamiento.

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 entero (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]

* target\_zeros — objetivo de las observaciones de la respuesta "0"
* target\_ones — objetivo de las observaciones de la respuesta "1"

Muestra en la pantalla los tamaños de las cuatro tablas almacenadas en las variables (ya en el precódigo).

import pandas as pd

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

features\_zeros = features\_train[target\_train == 0]

features\_ones = features\_train[target\_train == 1]

target\_zeros = target\_train[target\_train == 0]

target\_ones = target\_train[target\_train == 1]

print(features\_zeros.shape)

print(features\_ones.shape)

print(target\_zeros.shape)

print(target\_ones.shape)

Resultado

(37411, 196)

(584, 196)

(37411,)

(584,)

¡Es correcto!

Las observaciones positivas son escasas en comparación con las negativas, por lo que debes apegarte al pensamiento positivo a través de la pura tiranía de la voluntad... y el sobremuestreo.

2.

Duplica las observaciones de clase positivas y combínalas con las negativas.Utiliza la función pd.concat() para concatenar las tablas. Usa la documentación como ayuda.

Concatenamos las tablas y almacenamos el resultado en la variable features upsampled. Repite el proceso para el objetivo y declara la variable target\_upsampled.

El número de repeticiones se almacena en la variable repeat.

Muestra en la pantalla los tamaños de las nuevas variables (en precódigo).

## Pistas

La función concat() toma la lista de tablas que necesitas concatenar. Así es como debes escribir el código:

pd.concat([features\_zeros, features\_ones])

Si aplicas la multiplicación de listas, obtienes una tabla con observaciones positivas duplicadas:

pd.concat([features\_zeros] + [features\_ones] \* repeat)

import pandas as pd

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

features\_zeros = features\_train[target\_train == 0]

features\_ones = features\_train[target\_train == 1]

target\_zeros = target\_train[target\_train == 0]

target\_ones = target\_train[target\_train == 1]

repeat = 10

features\_upsampled = pd.concat([features\_zeros] + [features\_ones] \* repeat)

target\_upsampled = pd.concat([target\_zeros] + [target\_ones] \* repeat)# <escribe el código aquí >

print(features\_upsampled.shape)

print(target\_upsampled.shape)

Resultado

(43251, 196)

(43251,)

Es correcto!

¡Parece que el pensamiento positivo realmente puede ayudarnos a aumentar la rareza de la clase positiva!

3.

Mezcla los datos. Usa la función shuffle() en las variables features\_upsampled y targets\_upsampled. Devuelve el resultado de la operación anterior.

Escribe la función upsample() y pásale tres argumentos (ya casi está lista):

* features
* target
* repeat

La función devolverá las características y el objetivo después del *sobremuestreo*.

Llama a la función para los datos de entrenamiento. Muestra en la pantalla los tamaños de las muestras (en precódigo).

Pistas

## Pistas

Termina la función *upsample()*:

def upsample(features, target, repeat):

features\_zeros = features[target == 0]

features\_ones = features[target == 1]

*# < escribe el código aquí >*

return features\_upsampled, target\_upsampled

Usa este código para *shuffle*:

features\_upsampled, target\_upsampled = shuffle(features\_upsampled, target\_upsampled, random\_state=12345)

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.utils import shuffle

¡Es correcto!

Continúa practicando la fuerza de voluntad y el pensamiento positivo. Te ayudará cuando llegues al proyecto del curso.

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

features\_zeros = features\_train[target\_train == 0]

features\_ones = features\_train[target\_train == 1]

target\_zeros = target\_train[target\_train == 0]

target\_ones = target\_train[target\_train == 1]

repeat = 10

features\_upsampled = pd.concat([features\_zeros] + [features\_ones] \* repeat)

target\_upsampled = pd.concat([target\_zeros] + [target\_ones] \* repeat)

def upsample(features, target, repeat):

features\_zeros = features[target == 0]

features\_ones = features[target == 1]

return features\_upsampled, target\_upsampled

features\_upsampled, target\_upsampled = shuffle(features\_upsampled, target\_upsampled, random\_state=12345)# < añadir aleatorio >

features\_upsampled, target\_upsampled = upsample(

features\_train, target\_train, 10)

print(features\_upsampled.shape)

print(target\_upsampled.shape)

4.

Entrena el modelo *LogisticRegression* con los nuevos datos. Encuentra el valor *F1* y muéstralo en la pantalla (en precódigo).

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.utils import shuffle

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import f1\_score

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

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=12345

)

return features\_upsampled, target\_upsampled

features\_upsampled, target\_upsampled = upsample(

features\_train, target\_train, 10

)

model = LogisticRegression(solver='liblinear', random\_state=12345)

model.fit(features\_upsampled, target\_upsampled)

# Hacer predicciones con el conjunto de validación

predictions = model.predict(features\_valid)

# Calcular el valor F1

f1 = f1\_score(target\_valid, predictions)

print('F1:', f1\_score(target\_valid, predictions))

Resultado

F1: 0.13688212927756654

¡Es correcto!

¡Nuestra métrica está mejorando constantemente gracias al sobremuestreo! Esta técnica ayuda al modelo a reconocer y clasificar mejor las instancias de clase minoritaria, lo que finalmente aumenta nuestro valor F1.

# Submuestreo

### El submuestreo es una técnica que se utiliza para equilibrar las clases dentro de un conjunto de datos de entrenamiento. Al reducir la frecuencia de las observaciones de una clase predominante, podemos mejorar el rendimiento del modelo de aprendizaje automático al evitar que esté sesgado hacia la clase más común. ¿Cómo podemos hacer que las observaciones de una clase frecuente sean menos frecuentes en los datos?

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:

* División del conjunto de datos: separar el dataset de entrenamiento en dos grupos, uno para cada clase (positiva y negativa).
* Selección aleatoria: usa la función sample() para eliminar aleatoriamente una proporción de las observaciones de la clase predominante (negativa en este caso).
* Creación de una nueva muestra de entrenamiento: combina las observaciones restantes de la clase negativa con todas las observaciones de la clase positiva para forma un nuevo conjunto de entrenamiento.
* Mezcla de datos: si los datos no están bien mezclados, el modelo podría aprender patrones que no son realmente representativos del problema que se está abordando, sino que simplemente reflejan el orden en que los datos se presentaron durante el entrenamiento.

Para eliminar aleatoriamente algunos elementos de la tabla, utiliza la función sample(). Esta función requiere un parámetro llamado frac ('fraction' o fracción), que especifica la proporción de los elementos totales que quieres retener. Por ejemplo, frac=0.1 significa que la función seleccionará aleatoriamente el 10% de los elementos de la tabla original para formar una nueva muestra.

print(features\_train.shape)

features\_sample = features\_train.sample(frac=0.1, random\_state=12345)

print(features\_sample.shape)

(37995, 196)

(3800, 196)

Especifica random\_state=12345 para que tu código sea más fácil de revisar.

1.

Para realizar el *submuestreo*, escribe una función downsample() y pásale tres argumentos:

* features
* target
* fraction (fracción de observaciones negativas a mantener)

La función debe realizar el submuestreo y devolver tanto las características modificadas como las etiquetas. Antes de devolver estos datos, asegúrate de mezclarlos para evitar sesgos en el proceso de aprendizaje del modelo.

Llama a la función para los datos de entrenamiento con una *fracción* de 0.1. Muestra en la pantalla los tamaños de las muestras (en precódigo).

2.

Entrena el modelo *LogisticRegression* con los nuevos datos. Encuentra el valor *F1* y muéstralo en la pantalla (en precódigo).

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.utils import shuffle

# Cargar datos

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

# Definir características y etiquetas

target = data['Claim']

features = data.drop('Claim', axis=1)

# Dividir el conjunto de datos en entrenamiento y validación

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

# Definir la función de submuestreo

def downsample(features, target, fraction):

# Separar las características y etiquetas por clase

features\_zeros = features[target == 0]

features\_ones = features[target == 1]

target\_zeros = target[target == 0]

target\_ones = target[target == 1]

# Submuestrear las características y etiquetas de la clase mayoritaria (0)

features\_zeros\_downsampled = features\_zeros.sample(frac=fraction, random\_state=12345)

target\_zeros\_downsampled = target\_zeros.sample(frac=fraction, random\_state=12345)

# Combinar las observaciones submuestreadas de la clase 0 con todas las observaciones de la clase 1

features\_downsampled = pd.concat([features\_zeros\_downsampled, features\_ones])

target\_downsampled = pd.concat([target\_zeros\_downsampled, target\_ones])

# Mezclar los datos para evitar sesgos

features\_downsampled, target\_downsampled = shuffle(features\_downsampled, target\_downsampled, random\_state=12345)

return features\_downsampled, target\_downsampled

# Aplicar la función de submuestreo con una fracción de 0.1

features\_downsampled, target\_downsampled = downsample(features\_train, target\_train, 0.1)

# Mostrar los tamaños de las muestras

print(features\_downsampled.shape)

print(target\_downsampled.shape)  
  
Resultado

(4325, 196)

(4325,)

¡Es correcto!

Hemos reducido los datos a través de la fuerza de voluntad, el pensamiento positivo y el submuestreo. ¿Sientes que te estás moviendo sutilmente hacia el nirvana de la ciencia de datos?

2.

Entrena el modelo *LogisticRegression* con los nuevos datos. Encuentra el valor *F1* y muéstralo en la pantalla (en precódigo).

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.utils import shuffle

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import f1\_score

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

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=12345)]

+ [features\_ones]

)

target\_downsampled = pd.concat(

[target\_zeros.sample(frac=fraction, random\_state=12345)]

+ [target\_ones]

)

features\_downsampled, target\_downsampled = shuffle(

features\_downsampled, target\_downsampled, random\_state=12345

)

return features\_downsampled, target\_downsampled

features\_downsampled, target\_downsampled = downsample(

features\_train, target\_train, 0.1

)

model = LogisticRegression(solver='liblinear', random\_state=12345)

model.fit(features\_downsampled, target\_downsampled)

# Hacer predicciones con el conjunto de validación

predictions = model.predict(features\_valid)

# Calcular el valor F1

f1 = f1\_score(target\_valid, predictions)

# < escribe el código aquí >

print('F1:', f1\_score(target\_valid, predictions))

Resultado

F1: 0.13333333333333333

Es correcto!

¡Sí funciona! La calidad ha aumentado. La mejora no es espectacular pero no te desanimes: los resultados pueden variar para diferentes conjuntos de datos.

Capítulo 4/8

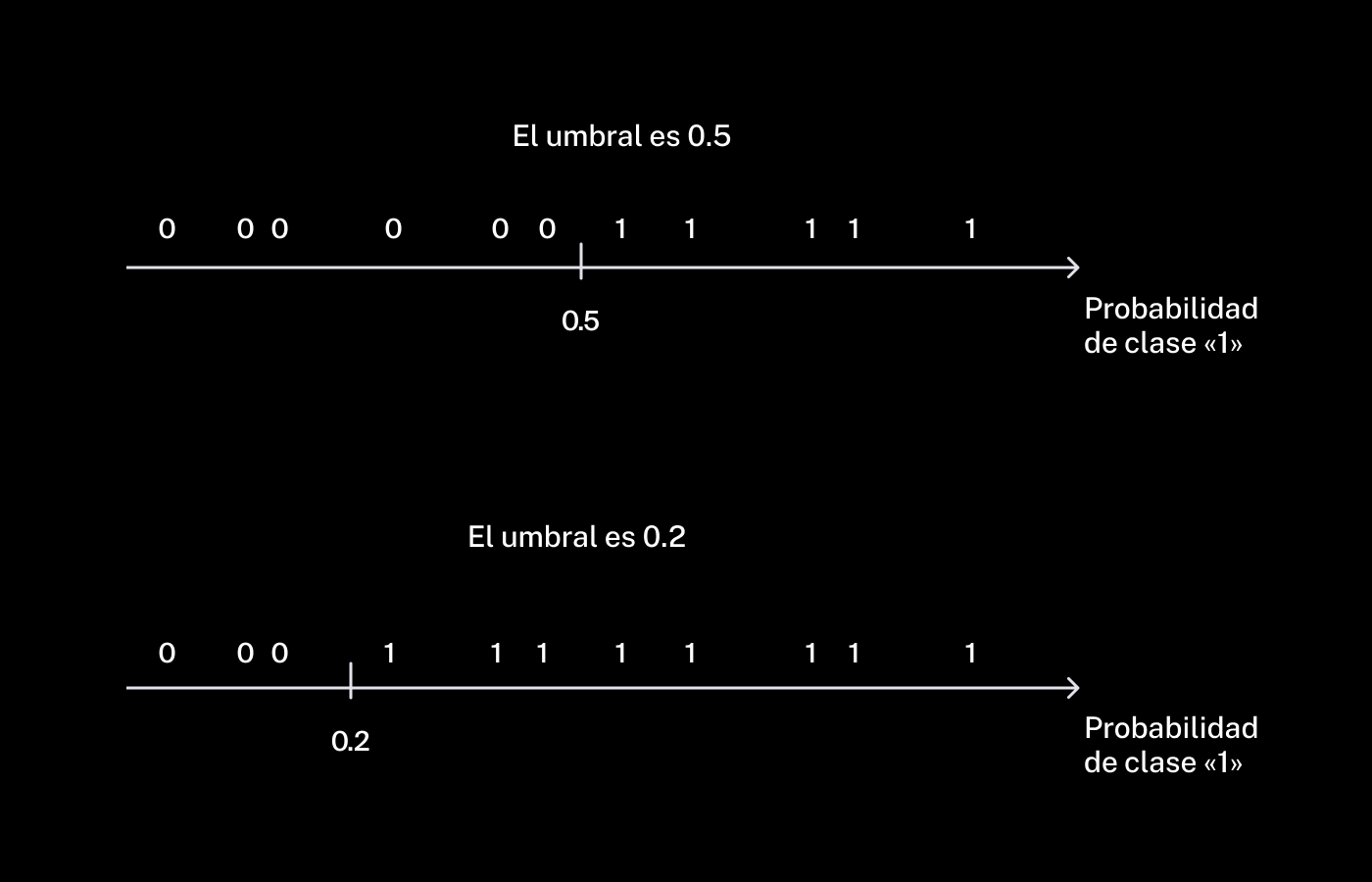
Clasificación desbalanceada

# Umbral de clasificación

### ¿Qué podemos hacer para optimizar el entrenamiento de la regresión logística? Deberíamos echar un vistazo a cómo funciona internamente.

Para determinar la respuesta, la regresión logística calcula la probabilidad de cada clase. Dado que solo tenemos dos clases (cero y uno), la probabilidad de la clase "1" es la que nos interesa. Esta probabilidad varía de cero a uno: si es mayor a 0.5, la observación se clasifica como positiva; si es menor, como negativa.

El punto de corte entre clasificaciones positivas y negativas se llama umbral. Por defecto es 0.5, pero ¿qué tal si lo cambiamos?



Pregunta

¿Cómo afecta la reducción del umbral a la precisión y a recall (sensibilidad)?

La precisión aumentará, recall disminuirá

La precisión disminuirá, recall aumentará

Habrá más respuestas positivas.

Nada cambiará

Tanto la precisión como recall aumentarán

# Ajuste de umbral

### A medida que cambiamos el valor del umbral, también veremos cómo cambian nuestras métricas.

En sklearn, esto se puede investigar utilizando la función predict\_proba(), que proporciona la probabilidad de que cada observación pertenezca a cada clase posible. Esta función toma las características de las observaciones como entrada y devuelve un array de probabilidades para cada clase.

probabilities = model.predict\_proba(features)

Así se verá el resultado si lo imprimimos: Este modelo genera dos probabilidades para cinco observaciones. La primera columna indica la probabilidad de clase negativa y la segunda indica la probabilidad de clase positiva (las dos probabilidades suman uno).

print(probabilities)

[[0.5795 0.4205]

[0.6629 0.3371]

[0.7313 0.2687]

[0.6728 0.3272]

[0.5086 0.4914]]

predict\_proba() también está disponible en *sklearn* para árboles de decisión y bosques aleatorios.

Ejercicio   1 / 2

1.

Encuentra las probabilidades de clase para la muestra de validación. Almacena los valores para las probabilidades de clase "1" en la variable probabilities\_one\_valid. Muestra en la pantalla los primeros cinco elementos de la variable (en precódigo).

## Pistas

Usa la función predict\_proba():

probabilities\_valid = *# < escribe el código aquí>*

probabilities\_one\_valid = probabilities\_ valid[:, 1]

Resultado

[0.01855917 0.009525 0.00419783 0.00157461 0.00956259]

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = LogisticRegression(random\_state=12345, solver='liblinear')

model.fit(features\_train, target\_train)

# Obtener las probabilidades de clase para la muestra de validación

probabilities\_valid = model.predict\_proba(features\_valid)

# Almacenar las probabilidades de la clase "1" en la variable probabilities\_one\_valid

probabilities\_one\_valid = probabilities\_valid[:, 1]

# Mostrar los primeros cinco elementos de probabilities\_one\_valid

print(probabilities\_one\_valid[:5])

¡Es correcto!

Conviértete en un héroe. Tolerancia cero. Zona Cero. Tienes tu propia lista con ceros: todas las probabilidades son cercanas a cero.

Capítulo 4/8 · Faltan 3 lecciones

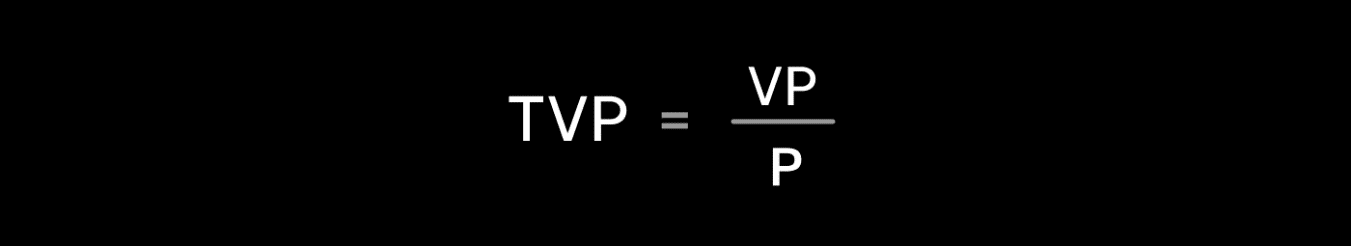
Clasificación desbalanceada

# TVP y TFP

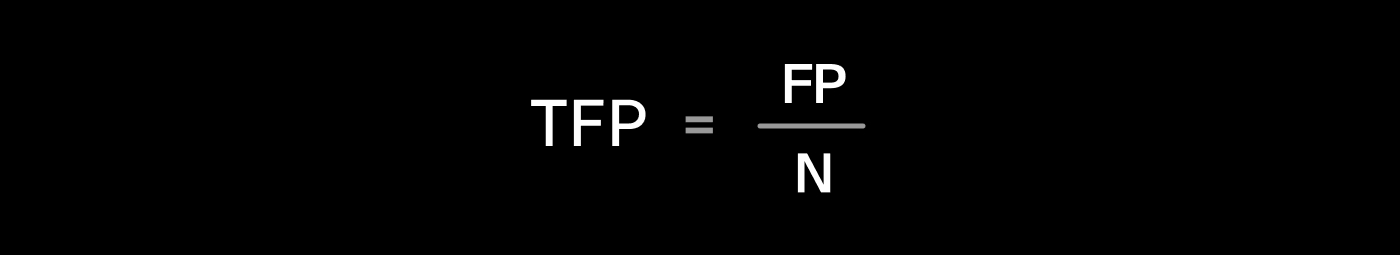
### No puedes calcular la precisión cuando no hay observaciones positivas. Consideremos las métricas que no implican dividir entre cero.

Antes de avanzar, definamos algunos términos importantes.

*Tasa de verdaderos positivos*, o *TVP*, es el resultado de las respuestas *VP* divididas entre todas las respuestas positivas. Aquí está la fórmula, donde *P* son todas las respuestas positivas:



La *Tasa de falsos positivos*, o *TFP*, es el resultado de las respuestas *FP* divididas entre todas las respuestas negativas. Se calcula usando una fórmula similar, donde *N* son todas las respuestas negativas:



En ambas métricas, los denominadores son valores fijos que no cambian con ajustes en el modelo, por lo que no existe riesgo de división entre cero.

Pregunta

¿Cómo afectará la disminución del umbral *TFP* en una regresión logística?

TFP disminuirá

TFP no cambiará

TFP aumentará

¡Correcto! El número de respuestas falsas positivas es proporcional al número de todas las respuestas negativas.

¡Lo has entendido bien!

Pregunta

¿Cómo afectará la disminución del umbral *TVP* en una regresión logística?

*TVP* disminuirá

TVP aumentará

¡Correcto! El número de respuestas verdaderas positivas es proporcional al número de todas las respuestas positivas.

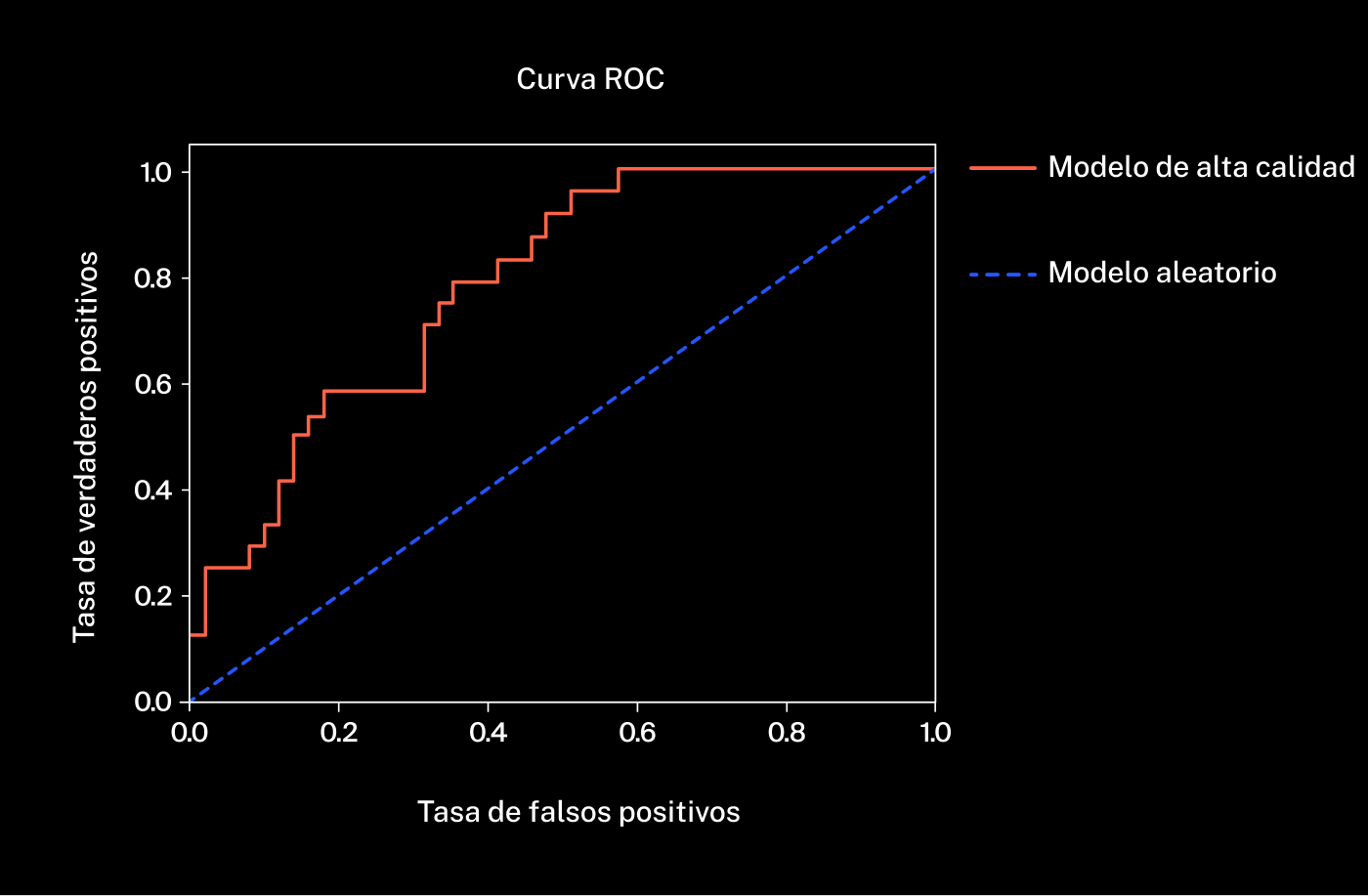
TVP no cambiará

# ¡Perfecto! Curva ROC

### Hemos observado un enfrentamiento entre la *Tasa de Verdaderos Positivos (TVP)* y la *Tasa de Falsos Positivos (TFP)*. Ahora, vamos a visualizar esta relación trazando la curva ROC.

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 llama la curva ROC (del inglés, Característica Operativa del Receptor,\* un término de la teoría del procesamiento de señales).

Para un modelo que siempre responde aleatoriamente, la curva *ROC* es una línea diagonal que va desde la esquina inferior izquierda hasta la esquina superior derecha. Cuanto más se aleje la curva ROC de esta línea diagonal hacia la esquina superior izquierda, mejor será el modelo, ya que indica una mayor relación *TVP-TFP*.



Para encontrar cuánto difiere nuestro modelo del modelo aleatorio, calculemos el valor AUC-ROC (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 *TFP*, valores *TVP* y los umbrales que superó.

fpr, tpr, thresholds = roc\_curve(target, probabilities)

Tracemos una gráfica para las variables *tfp* y *tvp.*

1.

Haz una curva *ROC* para la regresión logística y trázala en la gráfica. Sigue las instrucciones en el precódigo.

También agregamos la curva *ROC* del modelo aleatorio al precódigo.

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import roc\_curve

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = LogisticRegression(random\_state=12345, solver='liblinear')

model.fit(features\_train, target\_train)

probabilities\_valid = model.predict\_proba(features\_valid)

probabilities\_one\_valid = probabilities\_valid[:, 1]

fpr, tpr, thresholds = roc\_curve(target\_valid, probabilities\_one\_valid)# < escribe el código aquí >

plt.figure()

plt.plot(fpr, tpr)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

plt.xlabel('Tasa de falsos positivos')

plt.ylabel('Tasa de verdaderos positivos')

plt.title('Curva ROC')

plt.show()

# Curva ROC para modelo aleatorio (parece una línea recta)

plt.plot([0, 1], [0, 1], linestyle='--')

plt.show()  
  
Resultado

Gráfico, Gráfico de líneas, Gráfico de dispersión

Descripción generada automáticamenteGráfico, Gráfico de líneas

Descripción generada automáticamente

Es correcto!

¿Qué podemos decir de esta gráfica? Una línea recta no siempre es la mejor manera.La curva nos ayudará a encontrar la mejor proporción de *TFP* y *TVP* para nuestra tarea.

2.

Calcula el *AUC-ROC* para la regresión logística. Encuentra la función adecuada en la documentación de *sklearn*, así como la descripción de cómo funciona. Importa la función. Muestra en la pantalla el valor *AUC-ROC*.

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import roc\_curve, roc\_auc\_score

data = pd.read\_csv('/datasets/travel\_insurance\_us\_preprocessed.csv')

target = data['Claim']

features = data.drop('Claim', axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = LogisticRegression(random\_state=12345, solver='liblinear')

model.fit(features\_train, target\_train)

probabilities\_valid = model.predict\_proba(features\_valid)

probabilities\_one\_valid = probabilities\_valid[:, 1]

fpr, tpr, thresholds = roc\_curve(target\_valid, probabilities\_one\_valid)# < escribe el código aquí >

auc\_roc = roc\_auc\_score(target\_valid, probabilities\_one\_valid)

print(auc\_roc)  
  
Resultado

0.8222607565781999

Es correcto!

El resultado es mejor que el modelo aleatorio, pero todavía está lejos de ser perfecto (*AUC-ROC = 1).*

Sprint 9

Capítulo 4/8 · Última lección

Clasificación desbalanceada

# Conclusión

### ¡Ahora sabemos cómo luchar contra el desequilibrio de clases!

### En este capítulo aprendiste:

* Cómo usar el ajuste de peso de clase, el *sobremuestreo* y el *submuestreo* para mejorar las métricas de evaluación
* Qué es la *curva ROC* y cómo se puede calcular el valor *AUC-ROC*
* Cómo establecer los umbrales de clasificación para equilibrar la relación *TVP/TFP*

En el próximo capítulo veremos las métricas de regresión.

### Llévate esto contigo

Descárgate el resumen del capítulo y la hoja informativa para poder consultarlos cuando los necesites.

* [Resumen del capítulo: Clasificación desequilibrada](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_8/moved_Resumen_del_captulo_clasificacin_desequilibrad.pdf)
* [Hoja informativa: Clasificación desequilibrada](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_7/Additional_materials/Hoja_informativa_clasificacin_desequilibrada_ES.pdf?etag=954bf9adf1c85981ca1156b528f00b70)

Sprint 9

Capítulo 5/8

Métricas de regresión

# Introducción

### Veamos las métricas de regresión.

## Empezarás por:

* aprender sobre la métrica *R2* y cómo calcularla;
* aprender a evaluar la calidad del modelo usando la métrica *EAM*;
* descubrir las dos métricas clave, *EAM* y *RECM*.

## ¿Cuánto tiempo tomará?

Diez lecciones de 5-10 minutos cada una.

## Descripción del ejercicio

Entrena un modelo que prediga el tiempo de retraso de un vuelo.

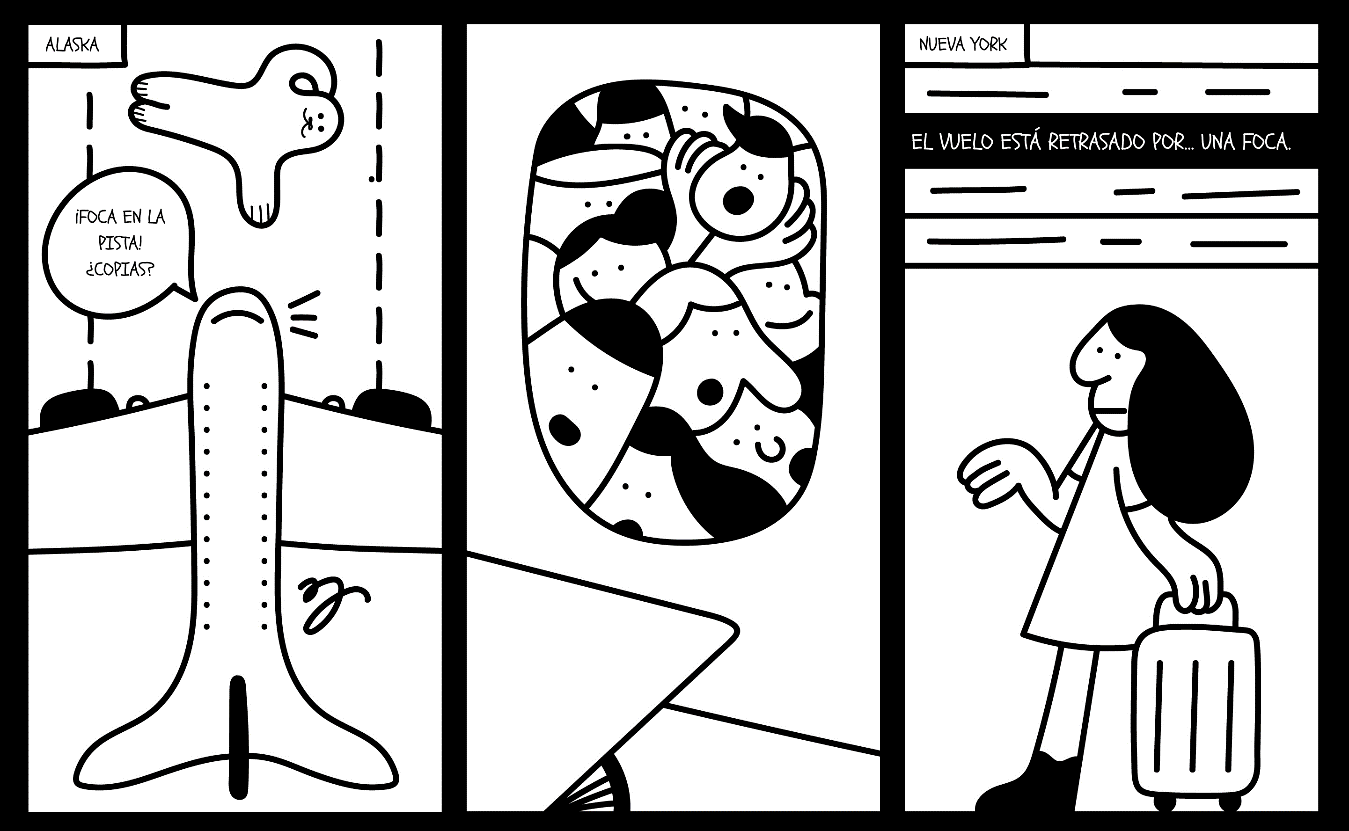
# Preparación de datos

### Veamos la tarea, carguemos los datos e investiguemos sus características.

Cada año crece la carga de infraestructura de las aerolíneas. Por ejemplo, en diciembre de 2018, el Aeropuerto Internacional de Bombay estableció un récord mundial con 1007 aviones que completaron el despegue y el aterrizaje ¡en 24 horas!

Los vuelos retrasados ponen en peligro el funcionamiento del aeropuerto, perjudicando los ingresos tanto del aeropuerto como de la aerolínea.

Para resolver este problema, entrenaremos un modelo de regresión para predecir los tiempos de retraso de los vuelos en minutos.



## Descripción de datos

Tenemos las siguientes características:

* Month: mes de vuelo
* Day: fecha del vuelo
* Day Of Week: día de la semana del vuelo
* Airline: nombre de la aerolínea
* Origin Airport Delay Rate: tasa de retraso de vuelo en el aeropuerto de origen
* Destination Airport Delay Rate: tasa de retraso del vuelo en el aeropuerto de destino
* Scheduled Time: tiempo de vuelo programado
* Distance: distancia del vuelo
* Scheduled Departure Hour: hora de salida programada (hora)
* Scheduled Departure Minute: hora de salida programada (minuto)

Objetivo:

* Arrival Delay: retraso de llegada (minuto)

1.

Carga los datos de flights.csv. Se debe mostrar lo siguiente:

* Tamaño de la tabla
* Primeras cinco filas de la tabla

Mira los datos.

import pandas as pd

# Cargar los datos

data = pd.read\_csv('/datasets/flights.csv')

# Mostrar el tamaño de la tabla

print(data.shape)

# Mostrar las primeras cinco filas de la tabla

print(data.head())

Resultado

(77909, 11)

Month Day ... Scheduled Departure Minute Arrival Delay

0 1 2 ... 55 -13.0

1 1 4 ... 30 -12.0

2 1 4 ... 30 189.0

3 1 3 ... 25 -7.0

4 1 5 ... 46 -4.0

[5 rows x 11 columns]

Es correcto!

Lo que obtienes:

* Muchos datos, casi 78 mil observaciones.
* Las características numéricas, excepto una que es categórica (*Airline*).
* Uno de los valores objetivo supera las tres horas. ¡Qué retraso!

2.

Codifica la característica categórica utilizando *OHE*. Estandariza las características numéricas. Muestra los tamaños de la tabla (en precódigo).

Cuando muestres los resultados en pantalla, no prestes atención al texto en color rojo. Está bien.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

data = pd.read\_csv('/datasets/flights.csv')

data\_ohe = pd.get\_dummies(data, drop\_first=True)# < aplica one hot encoding y evita la trampa de las variables ficticias encodifica los datos y evita la trampa de las variables ficticias >

target = data\_ohe['Arrival Delay']

features = data\_ohe.drop(['Arrival Delay'], axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

numeric = ['Day', 'Day Of Week', 'Origin Airport Delay Rate',

'Destination Airport Delay Rate', 'Scheduled Time', 'Distance',

'Scheduled Departure Hour', 'Scheduled Departure Minute']

scaler = StandardScaler()

scaler.fit(features\_train[numeric])

features\_train[numeric] = scaler.transform(features\_train[numeric])

features\_valid[numeric] = scaler.transform(features\_valid[numeric])# < transforma el conjunto de validación >

print(features\_train.shape)

print(features\_valid.shape)

Resultado

(58431, 22)

(19478, 22)

¡Es correcto!

Los datos están preparados.

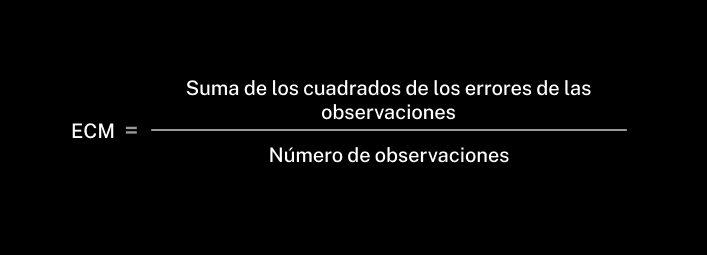
¡Listos para el despegue!

# EMC en una tarea de regresión

### ¿Qué métrica es la más adecuada para una tarea de regresión? ¡*EMC*! Entrenemos el modelo y verifiquemos su calidad.

Revisemos cómo calcular *ECM* (*error cuadrático medio*) y *RECM* (*raíz cuadrada del error cuadrático medio*).

Recuerda que en los ejercicios debéis referiros a los términos en inglés MSE y RMSE, que hacen referencia a ECM y RECM respectivamente.



Texto

Descripción generada automáticamente

1.

Carga los datos de /datasets/flights\_preprocessed.csv. Declara la variable predicted\_valid. Entrena la regresión lineal. Calcula el valor *ECM* para el conjunto de validación y guárdalo en la variable ecm.

Muestra los valores *ECM* y *RECM* (en precódigo).

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

data = pd.read\_csv('/datasets/flights\_preprocessed.csv')

target = data['Arrival Delay']

features = data.drop(['Arrival Delay'], axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = LinearRegression()

model.fit(features\_train, target\_train)

predicted\_valid = model.predict(features\_valid)# < escribe el código aquí >

mse = mean\_squared\_error(target\_valid, predicted\_valid)

print("MSE =", mse)

print("RMSE =", mse \*\* 0.5)  
  
Resultado

MSE = 2129.8240528555293

RMSE = 46.1500168240005

¡Es correcto!

1. ¿Es mucho para un *RECM*? ¡Vamos a realizar la prueba de cordura del modelo!

2.

Encuentra los valores de *ECM* y *RECM* para el modelo constante: esto predice el valor objetivo medio para cada observación. Almacena sus predicciones en la variable predicted\_valid.

Imprime los valores de *ECM* y *RECM* (en precódigo).

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

data = pd.read\_csv('/datasets/flights\_preprocessed.csv')

target = data['Arrival Delay']

features = data.drop(['Arrival Delay'], axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = LinearRegression()

model.fit(features\_train, target\_train)

predicted\_valid = model.predict(features\_valid)

mse = mean\_squared\_error(target\_valid, predicted\_valid)

print('Linear Regression')

print('MSE =', mse)

print('RMSE =', mse \*\* 0.5)

predicted\_valid = pd.Series(target\_train.mean(), index=target\_valid.index)

mse = mean\_squared\_error(target\_valid, predicted\_valid)# < escribe el código aquí >

print('Mean')

print('MSE =', mse)

print('RMSE =', mse \*\* 0.5)  
  
  
Resultado

Linear Regression

MSE = 2129.8240528555293

RMSE = 46.1500168240005

Mean

MSE = 2358.8874869200226

RMSE = 48.568379496540985

¡Es correcto!

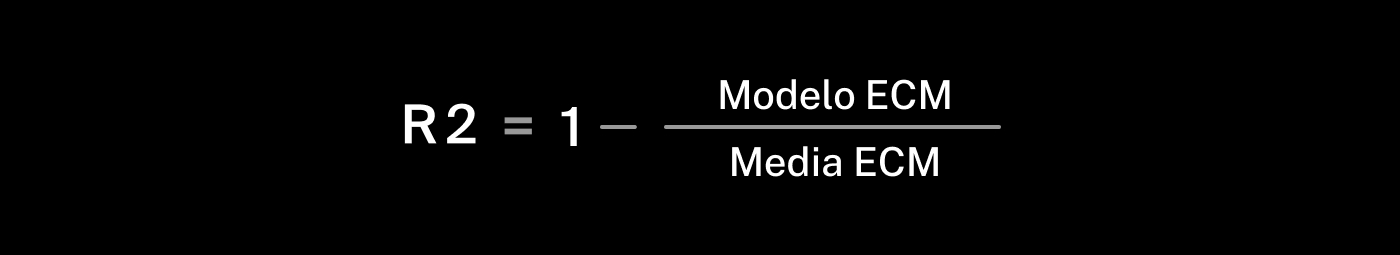
La diferencia de *RECM* es de solo dos minutos, pero todo es relativo: puedes cambiar el mundo en dos minutos.

# Coeficiente de determinación

### Para evitar comparar constantemente el modelo con la media, introduzcamos una nueva métrica. Esta se expresa en valores relativos, no en absolutos.

El coeficiente de determinación o la métrica R2 (*R-squared*) divide el *ECM* del modelo entre el *ECM* de la media y luego resta el valor obtenido de uno. Si la métrica aumenta, la calidad del modelo también mejora.

*R2* se calcula de la siguiente manera:



* *R2* es igual a uno solo si el ECM del modelo es cero. Dicho modelo predeciría perfectamente todas las respuestas.
* *R2* es cero: el modelo funciona tan bien como la media.
* Cuando *R2* es negativo, peor que usar solo la media para predecir.
* *R2* no puede tener valores mayores a uno porque esto indicaría que el modelo tiene un poder predictivo negativo, lo cual no es posible bajo la definición estándar de esta métrica.

Calcula el valor *R2* para la regresión lineal. Busca la función adecuada en la documentación de sklearn.metrics. Impórtala.

Muestra el resultado (en precódigo).

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score, mean\_squared\_error

data = pd.read\_csv('/datasets/flights\_preprocessed.csv')

target = data['Arrival Delay']

features = data.drop(['Arrival Delay'], axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = LinearRegression()

model.fit(features\_train, target\_train)

predicted\_valid = model.predict(features\_valid)

# Calcular MSE y RMSE

mse = mean\_squared\_error(target\_valid, predicted\_valid)

rmse = mse \*\* 0.5

print('R2 =', r2\_score(target\_valid, predicted\_valid)) # < escribe el código aquí >)  
  
Resultado

R2 = 0.09710497146204988

¡Es correcto!

No hay nada qué destacar, pero al menos el valor *R2* es mayor que cero. Eso es un comienzo.

# Maximización de R2

### *R2* es más fácil de entender que la RECM. Quedémonos con *R2* y escojamos el mejor modelo.

Crea un modelo con el valor *R2* más alto posible. Para resolver el ejercicio necesitamos que sea al menos 0.14.

Aquí tienes algunas sugerencias:

1. Encuentra el *R2* del modelo usando la función score(). Esta es la métrica predeterminada para los modelos de regresión en *sklearn*.

model = LinearRegression()

model.fit(features\_train, target\_train)

print(model.score(features\_valid, target\_valid))

0.09710497146204988

1. Elige la profundidad correcta del árbol. Comencemos con una pequeña cantidad de árboles. El número de árboles es proporcional a la calidad del modelo, pero también a la duración del entrenamiento, así que tenlo en mente.

for depth in range(1, 16, 1):

model = RandomForestRegressor(n\_estimators=20, max\_depth=depth, random\_state=12345)

model.fit(features\_train, target\_train)

*# < escribe el código aquí >*

Luego inicia el entrenamiento de bosque aleatorio con una gran cantidad de árboles:

model = RandomForestRegressor(n\_estimators=100,

max\_depth=*# < escribe el código aquí>, random\_state=12345)*

model.fit(features\_train, target\_train)

print(model.score(features\_train, target\_train))

print(model.score(features\_valid, target\_valid))

1. Entrenar un bosque de tal escala puede llevar unos minutos. En *Jupyter Notebook,* el tiempo de ejecución de la celda se mide con el comando %%time:

%%time

model = RandomForestRegressor(n\_estimators=100, random\_state=12345)

model.fit(features\_train, target\_train)

CPU times: user 48 s, sys: 928 ms, total: 48.9 s

Wall time: 54.9 s

El último texto Wall time se refiere al tiempo total transcurrido, medido como lo haría un reloj de pared, lo cual es lo que buscamos medir.

Crea un modelo con un valor *R2* de al menos 0.14. No escales las características. Usa estos datos: flights\_preprocessed.csv

Divide los datos en un conjunto de entrenamiento y un conjunto de validación para elegir el mejor modelo.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error

data = pd.read\_csv("/datasets/flights\_preprocessed.csv")

target = data["Arrival Delay"]

features = data.drop(["Arrival Delay"], axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = RandomForestRegressor(n\_estimators=100, max\_depth=11, random\_state=12345)

model.fit(features\_train, target\_train)

print("Configuración del modelo actual lograda:")

print("Valor R2 en un conjunto de entrenamiento", model.score(features\_train, target\_train))

print("Valor R2 en un conjunto de validación:", model.score(features\_valid, target\_valid))  
  
Resultado

Configuración del modelo actual lograda:

Valor R2 en un conjunto de entrenamiento 0.3784551354930391

Valor R2 en un conjunto de validación: 0.16447073084035702

¡Es correcto!

¿Es esto el desfase horario? La brecha de calidad entre el conjunto de entrenamiento y el conjunto de prueba es sustancial. El modelo está claramente sobreajustado. ¡Pero lo más importante es que hemos logrado el resultado!

# Error absoluto medio

### Vamos a explorar otra métrica de evaluación conocida como **EAM** (Error Absoluto Medio o “Mean Absolute Error” en Inglés). Es similar al MSE (Error Cuadrático Medio), pero no eleva los errores al cuadrado.

¡Basta de hablar! Escribamos esta nueva métrica con símbolos. Primero, definamos la notación estándar en *Data Science*:

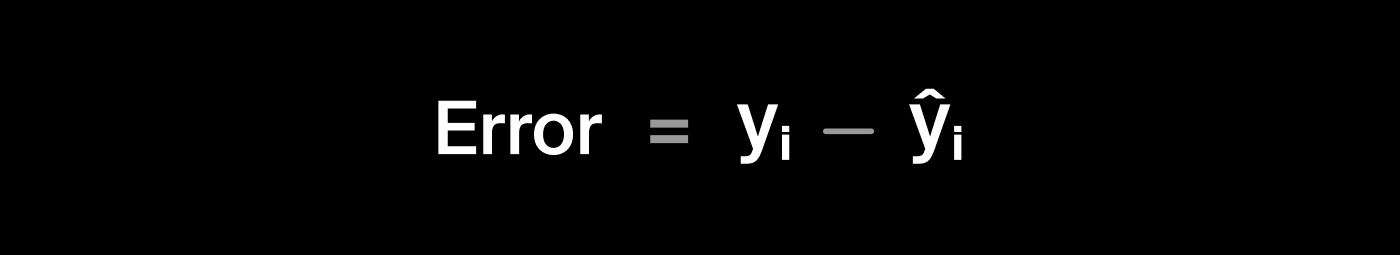


* Valor objetivo real para la observación *i* en el conjunto de datos (por ejemplo, un conjunto de prueba). El subíndice *i* indica el número de la observación.

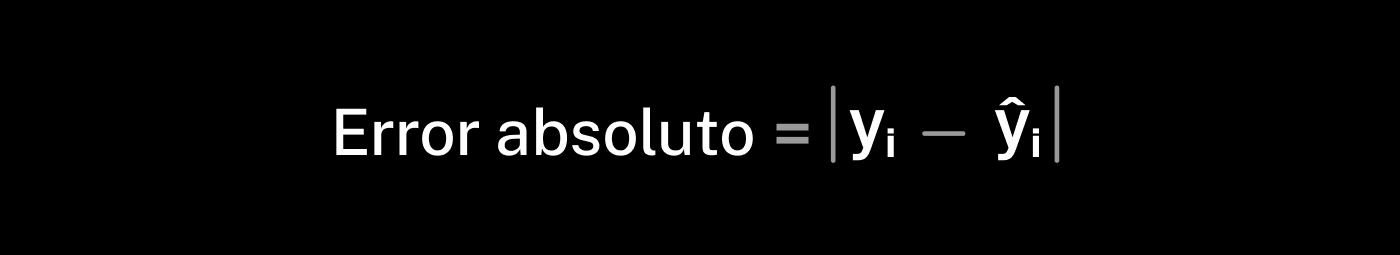


* Valor predicho de la variable objetivo para la observación *i* en el conjunto de datos. El símbolo "sombrero" o circunflejo sobre la *y* indica que se trata de una estimación y no del valor real.

El error de predicción se calcula como la diferencia entre el valor real y el valor predicho:



Para asegurarnos de que todas las diferencias se traten como valores positivos, utilizamos el valor absoluto del error:



Para recopilar los errores a lo largo del conjunto de datos, agreguemos la siguiente notación:

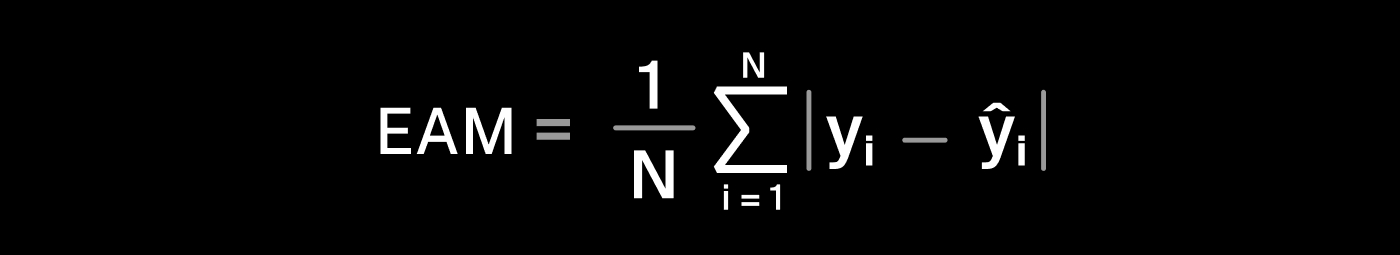


* El número de observaciones en la muestra.



* La suma de todas las observaciones del conjunto de datos (*i* varía en el rango de 1 a *N*).

Ahora podemos ver la fórmula para EAM o Error Absoluto Medio:



1.

Codifica la función mae() según la fórmula. Esta función las respuestas y predicciones correctas y devuelve el valor de error absoluto medio.

Considera que en Python el valor absoluto se calcula usando la función abs().

Prueba la función en el ejemplo en precódigo. Muestra el resultado en la pantalla.

## Pistas

En la fórmula, el número de serie comienza en 1, pero en una serie de pandas, el primer elemento tiene un índice cero.

Completa el código:

def mae(target, predictions):

error = 0

for i in range(target.shape[0]):

error += abs(*# < escribe el código aquí >)*

return error / target.shape[0]

import pandas as pd

def mae(target, predictions):

error = 0

for i in range(target.shape[0]):

error += abs(target[i] - predictions[i])

return error / target.shape[0]

target = pd.Series([-0.5, 2.1, 1.5, 0.3])

predictions = pd.Series([-0.6, 1.7, 1.6, 0.2])

print(mae(target, predictions))

Resultado

0.17500000000000004

¡Es correcto!

Ya codificamos la fórmula, volvamos a nuestro modelo

Por cierto, ¿sabías que EAM también significa experiencia artística multisensorial? ¡Eso es exactamente lo que obtienes cuando ves que el modelo funciona!

¡Es correcto!

Ya codificamos la fórmula, volvamos a nuestro modelo

Por cierto, ¿sabías que EAM también significa experiencia artística multisensorial? ¡Eso es exactamente lo que obtienes cuando ves que el modelo funciona!

2.

Calcula *EAM* para la regresión lineal. Encuentra la función apropiada en la documentación de *sklearn*. Impórtala. Muéstrala en la pantalla.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_absolute\_error# < escribe el código aquí >

data = pd.read\_csv('/datasets/flights\_preprocessed.csv')

target = data['Arrival Delay']

features = data.drop(['Arrival Delay'], axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = LinearRegression()

model.fit(features\_train, target\_train)

predicted\_valid = model.predict(features\_valid)

print(mean\_absolute\_error(target\_valid, predicted\_valid))

Resultado

27.43625097808585

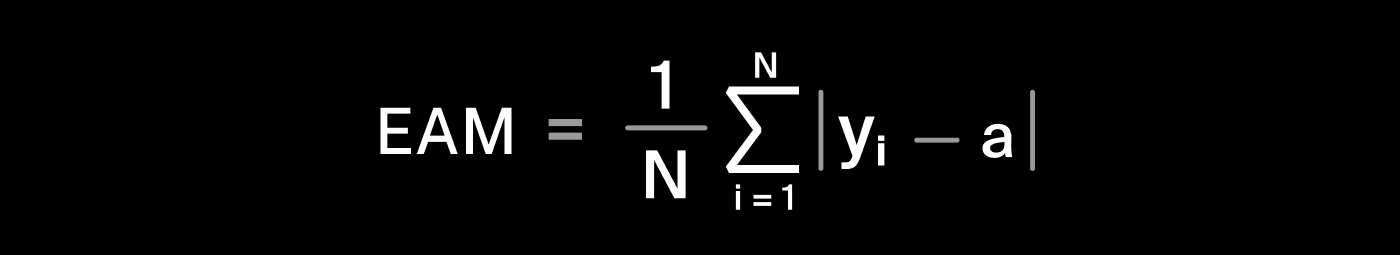
¡Es correcto!

¿Esto es mucho o poco? A diferencia de *RECM*, en *EAM* no se eleva al cuadrado ni se extrae la raíz. Por lo tanto, la interpretación de los valores es más fácil. Si el error absoluto medio del modelo es 27, entonces el error medio de cada observación es 27 minutos.

# Interpretación de EAM

### Cuando calculamos el Error Cuadrático Medio (ECM), a menudo usamos la media de los valores observados como una constante. ¿Podemos aplicar el mismo principio para el EAM?

En un modelo constante, el objetivo es seleccionar un valor que minimice el *EAM* de todas las predicciones. Necesitamos encontrar un valor constante, a, que resulte en el *EAM* más bajo posible. La fórmula para el *EAM* es la siguiente:



Curiosamente, el valor que minimiza el *EAM* es la mediana de los valores objetivo.

Ejercicio

Calcula el *EAM* utilizando la mediana como valor constante. El cálculo *EAM* para la regresión lineal se encuentra en el precódigo. Compara los valores.

Muestra el resultado en la pantalla.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_absolute\_error

data = pd.read\_csv('/datasets/flights\_preprocessed.csv')

target = data['Arrival Delay']

features = data.drop(['Arrival Delay'], axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model = LinearRegression()

model.fit(features\_train, target\_train)

predicted\_valid = model.predict(features\_valid)

print('Linear Regression')

print(mean\_absolute\_error(target\_valid, predicted\_valid))

print()

median\_value = target\_train.median()

predicted\_median = pd.Series(median\_value, index=target\_valid.index)

print('Median')

print(mean\_absolute\_error(target\_valid, predicted\_median))

Resultado

Linear Regression

27.436250978085834

Median

27.22281548413595

¡Es correcto!

La diferencia es de 1/5 minutos o 12 segundos. Ni siquiera lo notarías. Nuestro modelo lineal no fue muy útil.

# Minimización de EAM

### Intentemos minimizar el valor *EAM*.

Construye un modelo con *EAM* = 26.2: un minuto menos que la mediana. No es necesario pelear por las décimas.

Ejercicio

Construye un modelo con un valor *EAM* menor o igual a 26.2.

En la quinta lección de este capítulo ya aprendimos que *RandomForestRegressor* es una buena alternativa para la regresión lineal. Emplea la configuración de modelo que usaste en la lección 5 y verifica a cuál *EAM* puede llevar.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error

data = pd.read\_csv("/datasets/flights\_preprocessed.csv")

target = data['Arrival Delay']

features = data.drop(['Arrival Delay'], axis=1)

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

features, target, test\_size=0.25, random\_state=12345

)

model =RandomForestRegressor(n\_estimators=50, max\_depth=8, random\_state=12345)

# < escribe el código aquí >

model.fit(features\_train, target\_train)

predictions\_train = model.predict(features\_train)

predictions\_valid = model.predict(features\_valid)

print("Configuración del modelo actual lograda:")

print(

"Valor EAM en un conjunto de entrenamiento: ",

mean\_absolute\_error(target\_train, predictions\_train),

)

print(

"Valor EAM en un conjunto de validación: ",

mean\_absolute\_error(target\_valid, predictions\_valid),

)

Resultado

Configuración del modelo actual lograda:

Valor EAM en un conjunto de entrenamiento: 25.42062178810444

Valor EAM en un conjunto de validación: 25.90925992965721

Capítulo 5/8 · Faltan 2 lecciones

Métricas de regresión

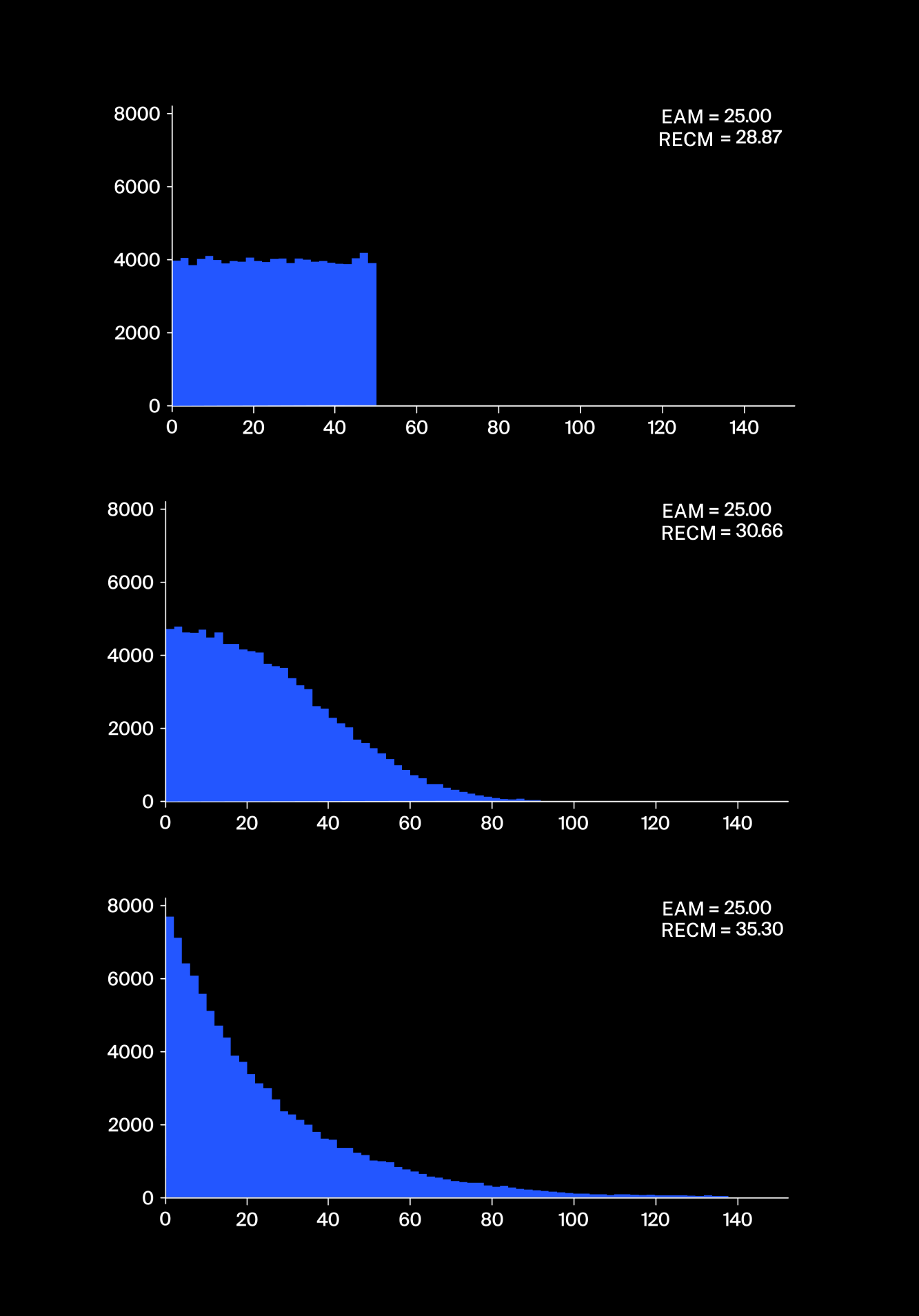
# El efecto de la dispersión en las métricas

### Veamos cómo los valores *EAM* y *RECM* dependen de la dispersión del objetivo.

A diferencia de *EAM*, la métrica *RECM* es bastante sensible a valores grandes: los errores significativos afectan fuertemente el valor final de *RECM*.

Echa un vistazo a los siguientes tres gráficos de distribución de errores (los valores de error se trazan a lo largo del eje horizontal y el número de errores está a lo largo del vertical).

* En el primer gráfico, el número de errores pequeños (con valores en el rango de 0 a 10) y el número de errores grandes (con valores de 40 y superiores) son iguales.
* En el segundo y tercer gráfico, la brecha aumenta gradualmente. Comienzan a aparecer errores muy grandes, aumenta el número de errores pequeños. Los valores de *EAM* no cambian: los errores grandes se equilibran con los errores pequeños. El *RECM*, por otro lado, aumenta, porque para *RECM* tomamos los errores al cuadrado y esto hace que la métrica sea más sensible a errores de gran magnitud, aunque dichos errores son pocos.



Ahora sabes que es posible cambiar el valor de una métrica sin cambiar otra.

Sprint 9

Capítulo 5/8 · Última lección

Métricas de regresión

# Conclusión

### Has resuelto el ejercicio de regresión construyendo diferentes modelos y has comprobado su calidad con un conjunto de nuevas métricas.

En este capítulo aprendiste:

* Cómo calcular el coeficiente de determinación y cómo se relaciona con el *ECM*
* Qué es la *métrica EAM*
* Cómo la dispersión del error afecta a los *EAM* y *RECM*

A continuación, analizaremos las habilidades blandas y luego tenemos el proyecto del curso.

### Llévate esto contigo

Descárgate el resumen del capítulo y la hoja informativa para poder consultarlos cuando los necesites.

* [Resumen del capítulo: Métricas de regresión](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_8/Resumen_del_capitulo_metricas_de_regresion.pdf?etag=53536c8252d9e9326b1e8a46061d7eac)
* [Hoja informativa: Métricas de regresión](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_sprint_8/esp/moved_Hoja_informativa_mtricas_de_regresin.pdf)

Sprint 9

Capítulo 6/8 · Faltan 2 lecciones

Habilidades socioemocionales

# Hechos, emociones y valoraciones

En esta frase "Me molesta que la estúpida plataforma en línea esté congelada", hay hechos, emociones y valoraciones: opciones:

* "La plataforma en línea está congelada" es un hecho
* "Me molesta" es una emoción
* "Estúpida plataforma en línea" es una valoración

Como principiante, es importante que distingas entre estos conceptos por dos razones:

1. Los profesionales son capaces de controlar sus emociones y son cuidadosos en sus valoraciones; prefieren utilizar hechos.
2. A veces, las personas toman decisiones influenciadas por sus emociones y esto puede afectar directamente a los datos que recibes.

Un hecho es algo que es objetivamente cierto. Puedes comprobar que la plataforma online está congelada.

Una emoción es una respuesta subjetiva. Un solo suceso (también un pensamiento) puede causar diferentes emociones: un estudiante puede sentirse molesto por la incapacidad de seguir trabajando de inmediato, mientras que otro puede sentir alegría de tomarse un descanso.

Una valoración es una opinión sobre un suceso, un tema o una persona, expresada de forma emotiva. No es una “valoración experta”. La valoración no tiene que ver con los hechos. La plataforma en línea no tiene un "cerebro" y no puede ser "estúpida". Es un conjunto de algoritmos que fallaron.

Pregunta

Selecciona todos los ejemplos donde hay valoraciones y hechos, pero no emociones:

Elige tantas como quieras

Las siguientes tres horas de trabajo fueron insoportablemente lentas.

"Tres horas de trabajo" es un hecho, "fueron insoportablemente lentas" es una valoración.

Me han asignado una nueva tarea de la nada.

"Nueva tarea" es un hecho, "de la nada" es una valoración personal sobre la forma en que recibiste la tarea.

La satisfacción con nuestro servicio cayó un 15% en el último mes, ¡esto es inaceptable!

"Un 15% en el último mes" es un hecho, "esto es inaceptable" es una valoración.

Me sorprende que nuevos clientes sigan usando nuestro servicio: ¡yo nunca lo usaría!

Mi jefe me acusa constantemente de descuido, ¡ya no puedo trabajar así!

¡Buen trabajo!

Solo recuerda analizar y estructurar tus pensamientos antes de formularlos:

* trabaja principalmente con hechos
* evita expresar emociones cuando las personas esperan una opinión experta de ti
* expón tus valoraciones y juicios, y aclara que es tu opinión personal

Sprint 9

Capítulo 6/8 · Última lección

Habilidades socioemocionales

# Habilidad para expresar una opinión

Pregunta

María ha estado trabajando durante tres semanas como data scientist en A-Tech Company, donde la gerencia acaba de cambiar. Se le asignó evaluar la métrica *EAM* para el modelo de machine learning. María estaba feliz porque la tarea no era difícil. Rápidamente encontró el número correcto y respondió el correo electrónico del gerente: "*EAM* es igual a 0.5":

Adivina qué pensó el gerente después de leer este correo electrónico: opciones:

¡Genial, *EAM* es inferior a 1!

0.5: ¿eso es bueno o malo?

El gerente esperaba obtener una conclusión e hipótesis, no solo un número. ¿Qué significa este número?

Si \* EAM\* es más de 0.1, es malo. Tenemos que arreglar la situación de inmediato.

¡Qué respuesta tan clara y concisa! María es definitivamente una buena analista.

¡Buen trabajo!

Los data scientist se involucran constantemente en diversas discusiones en el trabajo. Que sus conclusiones se tengan o no en cuenta depende de la claridad de la redacción y de la lógica de la justificación.

La mayoría de las veces estarás en contacto con representantes de dos grupos:

1. colegas involucrados en analytics o machine learning
2. personas que hacen solicitudes

Por lo general, con tus colegas puedes discutir tus inquietudes, compartir tareas complejas, buscar soluciones y solicitar retroalimentación sobre tu código. Es mejor que la discusión de trabajo sea constructiva:

* pídele a un colega que reserve tiempo para ti con anticipación
* indica claramente el propósito de la conversación
* describe los detalles, empezando por los hechos
* cuando expreses una opinión o valoración personal, aclara que es tu propio punto de vista.

La gente que hace solicitudes son gerentes o colegas de otras áreas (que por lo general conocen menos sobre los laberintos de la Data Science) No es apropiado hablar con ellos de métodos y problemas en los datos, pues lo que quieren son resultados y planear con base en eso. A manera de "algoritmo", esto es lo que harás:

* Vuelve a leer la solicitud original.
* Prepara una presentación sobre los resultados de acuerdo con esta solicitud; empieza con los números clave, luego expón conclusiones brevemente y:

a) si conoces los valores de la métrica del pasado, muestra la dinámica de sus cambios,

b) si es posible, asegúrate de que las conclusiones correspondan a la imagen general del negocio:

* En la presentación, habla sobre la lógica de las conclusiones más que de las herramientas y métodos de tu investigación.
* En tus hallazgos, no des instrucciones directas sobre qué hacer ahora (al menos mientras estés en un puesto junior). Esto está más allá de tu responsabilidad. Limítate a hacer recomendaciones.
* Durante la discusión, mantén una calma amistosa cuando se ponga a prueba tu lógica. Como un verdadero profesional debes estar listo para entender por qué se cuestionan tus conclusiones, y luego asegurarte de que tienes razón nuevamente o ver las áreas de oportunidad.

Pregunta

Imaginemos que María escribió una respuesta diferente a su gerente (es decir, la persona que realizó la solicitud), después de realizar una pequeña investigación de los eventos que habían ocurrido en los últimos dos meses. ¿Cuál es la redacción óptima? opciones:

El valor de *EAM* ha aumentado a 0.5. Encontré un problema: hubo fallas técnicas al cargar datos en la plataforma. Necesitamos arreglarlo, y luego los usuarios volverán.

Pregunté a mis colegas qué estaba pasando con la plataforma y descubrí que cuando nos mudamos a los nuevos servidores, algunos de los datos se guardaron con errores. Los usuarios escriben a soporte técnico y se quejan de errores. *EAM* ha subido a 0.5 y estamos perdiendo usuarios. Lo más importante ahora es restaurar la plataforma.

El modelo *EAM* es igual a 0.5 este mes. El mes pasado fue 0.4. Estamos viendo un aumento en la métrica. El cambio a los nuevos servidores ha causado problemas técnicos. Lo más probable es que el aumento en el valor de *EAM* se deba a datos corruptos. Si restauramos la copia de seguridad y volvemos a entrenar el modelo, la métrica debería volver a su valor original.

Sí, esta es la redacción más óptima de todas las opciones sugeridas. María presenta los números sin información o hechos superfluos. Números en las dinámicas sin información adicional y hechos sobre traslapes técnicos presentados. Y hay una hipótesis sin directivas en la última oración que es exactamente lo que se espera de un especialista junior.

*EAM* ha aumentado en 0.1, lo que está dentro del rango normal de fluctuación. No se detectaron problemas.

¡Felicidades! Has completado otro curso de la plataforma de entrenamiento. Ahora es el momento perfecto para probar tus habilidades y resolver un nuevo problema de machine learning. Realizarás este proyecto individualmente.

Cuando termines, envía tu trabajo al revisor del proyecto. Recibirás feedback dentro de las siguientes 48 horas. Después de eso, harás los cambios necesarios en tu trabajo y lo enviarás para una segunda revisión.

Por lo general, este proceso se repetirá varias veces hasta que recibas el visto bueno de la revisión y se aprueben todas las correcciones. Todo eso es parte del trabajo.

Tu proyecto se considerará completado una vez que el revisor del proyecto lo apruebe.

# Descripción del proyecto

Los clientes de Beta Bank se están yendo, cada mes, poco a poco. Los banqueros descubrieron que es más barato salvar a los clientes existentes que atraer nuevos.

Necesitamos predecir si un cliente dejará el banco pronto. Tú tienes los datos sobre el comportamiento pasado de los clientes y la terminación de contratos con el banco.

Crea un modelo con el máximo valor *F1* posible. Para aprobar la revisión, necesitas un valor *F1* de al menos 0.59. Verifica F1 para el conjunto de prueba.

Además, debes medir la métrica *AUC-ROC* y compararla con el valor *F1*.

### Instrucciones del proyecto

1. Descarga y prepara los datos. Explica el procedimiento.
2. Examina el equilibrio de clases. Entrena el modelo sin tener en cuenta el desequilibrio. Describe brevemente tus hallazgos.
3. Mejora la calidad del modelo. Asegúrate de utilizar al menos dos enfoques para corregir el desequilibrio de clases. Utiliza conjuntos de entrenamiento y validación para encontrar el mejor modelo y el mejor conjunto de parámetros. Entrena diferentes modelos en los conjuntos de entrenamiento y validación. Encuentra el mejor. Describe brevemente tus hallazgos.
4. Realiza la prueba final.

### Descripción de los datos

Puedes encontrar los datos en el archivo /datasets/Churn.csv file. [Descarga el conjunto de datos.](https://practicum-content.s3.us-west-1.amazonaws.com/datasets/Churn.csv)

Características

* *RowNumber*: índice de cadena de datos
* *CustomerId:* identificador de cliente único
* *Surname:* apellido
* *CreditScore:* valor de crédito
* *Geography:* país de residencia
* *Gender:* sexo
* *Age:* edad
* *Tenure:* período durante el cual ha madurado el depósito a plazo fijo de un cliente (años)
* *Balance:* saldo de la cuenta
* *NumOfProducts:* número de productos bancarios utilizados por el cliente
* *HasCrCard:* el cliente tiene una tarjeta de crédito (1 - sí; 0 - no)
* *IsActiveMember:* actividad del cliente (1 - sí; 0 - no)
* *EstimatedSalary:* salario estimado

Objetivo

* *Exited:* El cliente se ha ido (1 - sí; 0 - no)

# Evaluación del proyecto

Hemos definido los criterios de evaluación para el proyecto. Lee esto con atención antes de pasar al ejercicio.

Esto es lo que los revisores buscarán cuando evalúen tu proyecto:

* ¿Cómo preparaste los datos para el entrenamiento? ¿Procesaste todos los tipos de características?
* ¿Explicaste los pasos de preprocesamiento lo suficientemente bien?
* ¿Cómo investigaste el equilibrio de clases?
* ¿Estudiaste el modelo sin tener en cuenta el desequilibrio de clases?
* ¿Qué descubriste sobre la investigación del ejercicio?
* ¿Dividiste correctamente los datos en conjuntos?
* ¿Cómo trabajaste con el desequilibrio de clases?
* ¿Utilizaste al menos dos técnicas para corregir el desequilibrio?
* ¿Realizaste correctamente el entrenamiento, la validación y las pruebas finales del modelo?
* ¿Qué tan alto es tu valor *F1*?
* ¿Examinaste los valores *AUC-ROC*?
* ¿Mantuviste la estructura del proyecto y el código limpio?

Ya tienes las hojas informativas y los resúmenes de capítulos, tienes todo para continuar con el proyecto.

¡Buena suerte!

DecisionTreeClassifier

La mejor profundidad es: 9

La exactitud del mejor modelo en el conjunto de pruebas es de : 0.8665

La matriz de confusion para el modelo de arbol de clasificación es:

[[1453 105]

[ 231 211]]

La proporcion de respuestas VP(verdaderas positivas) es: 0.47737556561085975

La cantidad de respuestas negativas encontro el modelo mientras buscaba respuestas positivas es 0.6677215189873418

El valor de la metrica F1 es: 0.5259067357512953

RandomForestClassifier

La mejor profundidad es: 1

La exactitud del mejor modelo en el conjunto de pruebas con n\_estimators(estimadores) 20) es de : 0.8715

La matriz de confusion para el modelo de arbol de clasificación es:

[[1553 5]

[ 375 67]]

La proporcion de respuestas VP(verdaderas positivas) es: 0.1515837104072398

La cantidad de respuestas negativas encontro el modelo mientras buscaba respuestas positivas es 0.9305555555555556

El valor de la metrica F1 es: 0.5259067357512953

LogisticRegressio

La exactitud del mejor modelo en el conjunto de pruebas es de : 0.8715

La matriz de confusion para el modelo de arbol de clasificación es:

[[1525 33]

[ 387 55]]

La proporcion de respuestas VP(verdaderas positivas) es: 0.1244343891402715

La cantidad de respuestas negativas encontro el modelo mientras buscaba respuestas positivas es 0.625

El valor de la metrica F1 es: 0.20754716981132076

Sprint 9

Capítulo 8/8 · Faltan 2 lecciones

Conclusión

# Conclusión

¡Felicidades! Has completado el curso "Aprendizaje supervisado".

Aprendiste lo siguiente:

* Cómo trabajar con nuevas métricas de evaluación como *precisión*, *recall*, valor *F1* y *AUC-ROC*.
* Cómo ajustar los pesos de clase, el submuestreo, el sobremuestreo y los umbrales para las tareas de clasificación.
* Evaluar la calidad de un modelo utilizando las métricas *R2* y *EAM* en las tareas de regresión.

### ¿Y ahora qué?

El próximo curso estará dedicado al aprendizaje automático para empresas. Aprenderás cómo implementar tus modelos en escenarios del mundo real, cómo elegir las métricas de evaluación correctas en función de los valores objetivo del negocio y cómo realizar pruebas A/B.

Después de este sprint serás capaz de:

* Conectar las métricas de negocio con las métricas de ML;
* Distinguir entre métricas fuera de línea y métricas en línea;
* Usar bootstrap para calcular intervalos de confianza y métricas comerciales;
* Recopilar datos para evitar la fuga de datos y mejorar la confiabilidad de la evaluación del modelo.

Para tu proyecto final, construirás un modelo que ayudará a elegir la región con el mayor margen de beneficio para la empresa minera. Analizarás los beneficios y riesgos potenciales utilizando la técnica bootstrapping.

¿Cuánto tiempo llevará?

El siguiente sprint será de una dificultad media, y completarlo te tomará unas 30-50 horas.