Resumen del capítulo: Mejora del modelo

Datasets de validación

Para que el control de calidad sea fiable, necesitamos un dataset de **validación**.

El dataset de validación se separa del dataset fuente antes de que se entrene el modelo. De otro modo, el modelo sabría todas las respuestas antes de aprender del conjunto de entrenamiento. La validación muestra cómo se comporta el modelo en el campo y ayuda a revelar si hay sobreajuste.

La parte de los datos que se va a asignar al conjunto de validación depende del número de observaciones y características, así como de la variación de los datos. Estos son los dos escenarios más comunes:

1) El conjunto de prueba existe (o existirá en el futuro cercano), pero no está disponible por el momento. La proporción ideal es de 3:1. Esto significa que un 75 % es para el conjunto de entrenamiento y un 25 % es para el conjunto de validación. Este escenario se usará en nuestra plataforma de entrenamiento en línea.

2) El conjunto de prueba no existe. En ese caso, los datos fuente deben dividirse en tres partes: entrenamiento, validación y prueba. Usualmente, el tamaño del conjunto de validación y del conjunto de prueba son iguales. Este escenario nos da una proporción de 3:1:1 (un 60 % para el conjunto de entrenamiento y un 20 % para los conjuntos de validación y prueba).

División de los datos en dos conjuntos:

Para este propósito, Scikit-learn tiene una función especial llamada **train_test_split()**. Esta función puede dividir cualquier conjunto de datos en dos, y se denomina así porque suele utilizarse para dividir los conjuntos en conjuntos de entrenamiento y de prueba. Pero en nuestro caso, vamos a recurrir a esta función a fin de obtener un conjunto de entrenamiento y otro de validación.

```
from sklearn.model_selection import train_test_split
```

Antes de dividir, necesitamos establecer dos parámetros:

- **Nombre del dataset** que vamos a dividir.
- **Tamaño del conjunto de validación** (`test_size`). El tamaño se expresa con un decimal entre 0 y 1 que representa una fracción del dataset fuente. En este caso, tenemos `test_size=0.25` porque queremos trabajar con el 25 % del conjunto fuente.

La función `train_test_split()` devuelve dos conjuntos de datos, el de entrenamiento y el de validación.

```
df_train, df_valid = train_test_split(df, test_size=0.25, random_state=54321)
```

Nota: podemos asignar cualquier valor a `random_state` excepto `None`.

Hiperparámetros

El modelo adquiere todos los parámetros del conjunto de entrenamiento. Además de los parámetros de modelo regulares, tenemos **hiperparámetros**. Estos son configuraciones para algoritmos de aprendizaje. Es necesario especificarlos antes del entrenamiento.

Por ejemplo, en el árbol de decisión, los hiperparámetros son:

— **max_depth:** que es la profundidad máxima del árbol.

— **criterion:** que es el criterio de división.

— **min_samples_split:** que prohíbe crear nodos que no contengan suficientes observaciones del conjunto de entrenamiento.

— **min_samples_leaf:** las hojas son los nodos más bajos que contienen las respuestas y que dejan de dividir los datos. Este hiperparámetro evita que el algoritmo añada nodos hoja que no tengan suficientes observaciones del conjunto de entrenamiento.

Nuevos modelos: bosque aleatorio

Probemos con un nuevo algoritmo de aprendizaje llamado bosque aleatorio. Este algoritmo entrena una gran cantidad de árboles independientes y toma una decisión mediante el voto. Un bosque aleatorio ayuda a mejorar los resultados y a evitar el sobreajuste.

En la librería scikit-learn, puedes encontrar **RandomForestClassifier** que es un algoritmo de bosque aleatorio. Impórtalo desde el módulo **ensemble**:

```
from sklearn.ensemble import RandomForestClassifier
```

Para determinar el número de árboles en el bosque, usaremos el hiperparámetro `n_estimators` *(número de estimadores).* La calidad del resultado final y la duración del entrenamiento son estrictamente proporcionales al número de árboles.

```
model = RandomForestClassifier(random_state=54321, n_estimators=3)
```

Regresión logística

Si incrementamos el valor del **hiperparámetro `n_estimators`, el modelo se vuelve voluminoso y el proceso de entrenamiento se hace lento, lo que no es bueno. Pero si mantenemos el número de árboles bajo, los resultados no mejorarán. Así que eso también es malo.

Probemos con la **regresión logística**. Aunque el nombre sugiere un problema de regresión, sigue siendo un algoritmo de clasificación.

Para predecir la clase de un apartamento, la regresión logística hace lo siguiente:

- En primer lugar, decide a qué clase se acerca la observación.
- En función de la respuesta, elige la clase: si el resultado del cálculo es positivo, entonces será "1" (precios altos), mientras que si es negativo, será "0" (precios bajos).

Hay solo unos cuantos parámetros en la regresión logística. El modelo no será capaz de memorizar ninguna de las características de la fórmula, por lo que la probabilidad de sobreajuste será baja.

El modelo **LogisticRegression** se encuentra en el módulo *sklearn.linear_model* de la librería *sklearn*. Impórtalo:

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(random_state=54321)
```