Resumen del capítulo: Calidad del modelo

Aleatoriedad en algoritmos de aprendizaje

El hecho de aplicar un carácter aleatorio a los algoritmos de aprendizaje refuerza en gran medida la capacidad del modelo para descubrir relaciones en los datos. El barajado introduce aleatoriedad en el algoritmo de aprendizaje. Una computadora no puede generar números verdaderamente aleatorios. Utiliza **generadores de números pseudoaleatorios** que crean secuencias que parecen aleatorias.

Los números aleatorios no son tan simples. Son imprevisibles. Los generadores de números pseudoaleatorios pueden configurarse de manera que sus resultados sean siempre los mismos. Todo lo que tienes que hacer para añadir pseudoaleatoriedad al crear un algoritmo de aprendizaje es especificar el **parámetro `random_state`:

```
# especifica un estado aleatorio (número)
model = DecisionTreeClassifier(random_state=54321)

# entrena al modelo de la misma manera que antes
model.fit(features, target)
```

Si estableces `random_state` en `None` (que es su valor por defecto), la pseudoaleatoriedad será siempre diferente.

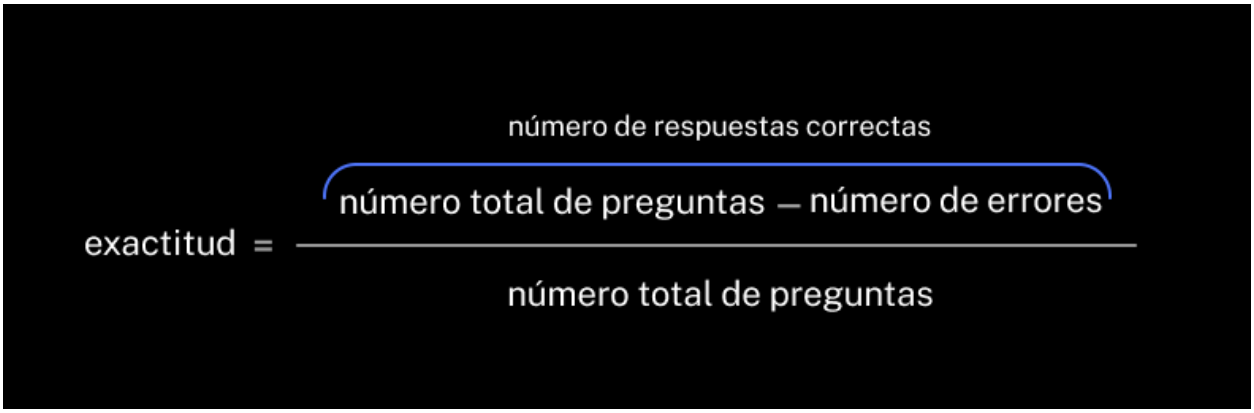
Dataset de prueba

Para probar si nuestro modelo hace predicciones precisas incluso a la hora de enfrentarse a nuevos datos, vamos a utilizar un nuevo conjunto de datos. Ese será el **conjunto de datos de prueba**.

Exactitud

La relación entre el número de respuestas correctas y el número total de preguntas (es decir, el tamaño del conjunto de datos de prueba) se denomina **exactitud (accuracy).**

Para calcular la *exactitud*, utiliza esta fórmula:



Métricas de evaluación

Las **métricas de evaluación** se utilizan para medir la calidad de un modelo y se pueden expresar numéricamente. Ya has encontrado una de estas métricas: *exactitud*. Hay otras, por ejemplo:

- ***Precisión*** (precision) toma todos los apartamentos que el modelo consideró caros (están marcados como "1") y calcula qué fracción de ellos era realmente costosa. Los apartamentos no reconocidos por el modelo se ignoran.
- ***Recall*** (sensibilidad) toma todos los apartamentos que son realmente caros y calcula qué fracción de ellos reconoció el modelo. Los apartamentos que fueron reconocidos por el modelo por error se ignoran.

Siempre asegúrate de que tu modelo funcione **mejor que la casualidad**, es decir, realiza una **prueba de cordura**.

Métricas de evaluación en Scikit-Learn

Puedes encontrar las funciones métricas de la librería scikit-learn en el módulo **metrics**. Utiliza la función **accuracy_score()** para calcular la *exactitud*.

```
from sklearn.metrics import accuracy_score
```

La función toma dos argumentos (respuestas correctas y predicciones del modelo) y devuelve el valor de *exactitud*.

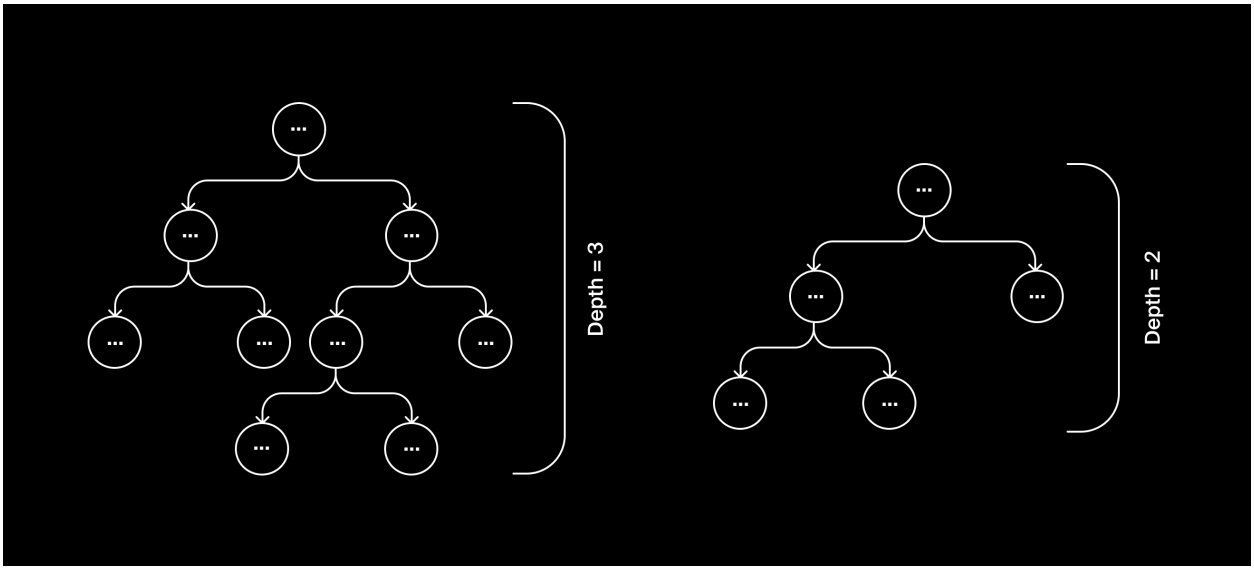
```
accuracy = accuracy_score(target, predictions)
```

Subajuste y sobreajuste

El modelo no tuvo problemas con los ejemplos del conjunto de entrenamiento, pero se atascó en el conjunto de pruebas. Esto representa un síntoma de **sobreajuste**. El efecto contrario se llama **subajuste**. Ocurre cuando la *exactitud* es baja y aproximadamente igual tanto para el conjunto de entrenamiento como para el de prueba.

No siempre es posible evitar el sobreajuste o el subajuste. Cuando mejoras en uno de estos, aumenta el riesgo del otro.

Revisa este ejemplo de excelente ajuste de un algoritmo de entrenamiento. ¿Cómo afecta al equilibrio entre el sobreajuste y el subajuste? **La profundidad del árbol (altura)** es la cantidad máxima de condiciones desde la "parte superior" del árbol hasta la respuesta final, según el número de transiciones de nodo a nodo.



La profundidad del árbol en *sklearn* puede establecerse mediante el parámetro `max_depth`:

```
# especifica la profundidad (ilimitado por defecto)
model = DecisionTreeClassifier(random_state=54321, max_depth=3)

model.fit(features, target)
```

Experimentos con el árbol de decisión

Para guardar el modelo entrenado en el formato correcto, utiliza la función *dump* de esta librería.

```
# guarda el modelo
# el primer argumento es el modelo
# el segundo argumento es la ruta al archivo

from joblib import dump

joblib.dump(model, 'model.joblib')
```

Puedes abrir y ejecutar el modelo usando la función *load*.

```
import joblib

# un argumento es la ruta al archivo
# un valor de retorno es el modelo
model = joblib.load('model.joblib')
```

-