

## Práctica

El objetivo de este ejercicio es evaluar el impacto de usar una codificación entera arbitraria para variables categóricas junto con un modelo de clasificación lineal como la Regresión Logística.

Para ello, intentemos usar `OrdinalEncoder` para preprocessar las variables categóricas. Este procesador se ensambla en un pipeline con `LogisticRegression`. El rendimiento de generalización del pipeline se puede evaluar mediante validación cruzada y luego comparar con el puntaje obtenido al usar `OneHotEncoder` o con algún otro puntaje de referencia.

Primero, cargamos el conjunto de datos.

```
import pandas as pd

adult_census = pd.read_csv("adult.csv")

target_name = "income"
target = adult_census[target_name]
data = adult_census.drop(columns=[target_name, "education.num"])
```

En el notebook previo, utilizamos `sklearn.compose.make_column_selector` para seleccionar automáticamente las columnas con un tipo de dato específico (también llamado `dtype`). Aquí, usamos este selector para obtener solo las columnas que contienen cadenas (columnas con `dtype object`), que corresponden a las características categóricas en nuestro conjunto de datos.

```
from sklearn.compose import make_column_selector as selector

categorical_columns_selector = selector(dtype_include=object)
categorical_columns = categorical_columns_selector(data)
data_categorical = data[categorical_columns]

data_categorical.head()
```

|   | workclass | education    | marital.status | occupation        | relationship  | race          | sex    | native.country |
|---|-----------|--------------|----------------|-------------------|---------------|---------------|--------|----------------|
| 0 | ?         | HS-grad      | Widowed        |                   | ?             | Not-in-family | White  | Female         |
| 1 | Private   | HS-grad      | Widowed        | Exec-managerial   | Not-in-family | White         | Female | United-States  |
| 2 |           | Some-college | Widowed        |                   | ?             | Unmarried     | Black  | Female         |
| 3 | Private   | 7th-8th      | Divorced       | Machine-op-inspct | Unmarried     | White         | Female | United-States  |
| 4 | Private   | Some-college | Separated      | Prof-specialty    | Own-child     | White         | Female | United-States  |

Next steps: [Generate code with data\\_categorical](#) [New interactive sheet](#)

```
data_categorical['workclass'].value_counts()
```

| workclass        | count |
|------------------|-------|
| Private          | 22696 |
| Self-emp-not-inc | 2541  |
| Local-gov        | 2093  |
| ?                | 1836  |
| State-gov        | 1298  |
| Self-emp-inc     | 1116  |
| Federal-gov      | 960   |
| Without-pay      | 14    |
| Never-worked     | 7     |

`dtype: int64`

Notarás que `workclass` y `occupation` tienen muchos valores desconocidos: ?. Elimina esas dos características.

```
data_categorical = data_categorical.drop(columns=['workclass', 'occupation'])
data_categorical.head()
```

|   | education    | marital.status | relationship  | race  | sex    | native.country | grid icon |
|---|--------------|----------------|---------------|-------|--------|----------------|-----------|
| 0 | HS-grad      | Widowed        | Not-in-family | White | Female | United-States  | info icon |
| 1 | HS-grad      | Widowed        | Not-in-family | White | Female | United-States  |           |
| 2 | Some-college | Widowed        | Unmarried     | Black | Female | United-States  |           |
| 3 | 7th-8th      | Divorced       | Unmarried     | White | Female | United-States  |           |
| 4 | Some-college | Separated      | Own-child     | White | Female | United-States  |           |

Next steps: [Generate code with data\\_categorical](#) [New interactive sheet](#)

Define un pipeline de scikit-learn compuesto por un `OrdinalEncoder` y un clasificador `LogisticRegression`.

Dado que `OrdinalEncoder` puede generar errores si encuentra una categoría desconocida en el momento de la predicción, puedes establecer los parámetros `handle_unknown="use_encoded_value"` y `unknown_value`. Puedes consultar la [documentación de scikit-learn](#) para obtener más detalles sobre estos parámetros.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import OrdinalEncoder
from sklearn.linear_model import LogisticRegression

model = make_pipeline(
    OrdinalEncoder(handle_unknown="use_encoded_value", unknown_value=-1),
    LogisticRegression(max_iter=1000, random_state=42)
)
```

Tu modelo ya está definido. Evalúalo utilizando validación cruzada con `sklearn.model_selection.cross_validate`.

Nota

Ten en cuenta que si ocurre un error durante la validación cruzada, `cross_validate` emitirá una advertencia y devolverá NaN (Not a Number) como puntajes. Para hacer que genere una excepción estándar de Python con un traceback, puedes pasar el argumento `error_score="raise"` en la llamada a `cross_validate`. Se generará una excepción en lugar de una advertencia ante el primer problema encontrado y `cross_validate` se detendrá de inmediato en lugar de devolver valores NaN. Esto es particularmente útil cuando se desarrollan pipelines complejos de aprendizaje automático.

```
from sklearn.model_selection import cross_validate

cv_results = cross_validate(
    model,
    data_categorical,
    target,
    cv=5,
    error_score="raise"
)

print("Resultados de validación cruzada:")
print(cv_results)

Resultados de validación cruzada:
{'fit_time': array([0.21720672, 0.25177026, 0.20239139, 0.2458303 , 0.25498843]), 'score_time': array([0.03248453, 0.02975917, 0.02}
```

```
scores = cv_results["test_score"]

print(f"The accuracy is: {scores.mean():.3f} ± {scores.std():.3f}")

The accuracy is: 0.755 ± 0.001
```

Ahora, compara el rendimiento de generalización del modelo anterior con un nuevo modelo donde, en lugar de usar un `OrdinalEncoder`, utilizamos un `OneHotEncoder`. Repite la evaluación del modelo utilizando validación cruzada. Compara el puntaje de ambos modelos y concluye sobre el impacto de elegir una estrategia de codificación específica al usar un modelo lineal.

```
from sklearn.preprocessing import OneHotEncoder

model_onehot = make_pipeline(
    OneHotEncoder(handle_unknown="ignore"),
    LogisticRegression(max_iter=1000, random_state=42)
)

cv_results_onehot = cross_validate(
    model_onehot,
    data_categorical,
    target,
    cv=5,
    scoring="accuracy",
    error_score="raise"
)

print("== Resultados con OneHotEncoder ==")
scores_onehot = cv_results_onehot["test_score"]
```

```
== Resultados con OneHotEncoder ==
```

```
print(f"Puntajes de validación cruzada: {scores_onehot}")
print(f"Accuracy: {scores_onehot.mean():.3f} ± {scores_onehot.std():.3f}")
```

```
Puntajes de validación cruzada: [0.83816981 0.81265356 0.81418919 0.82232801 0.81603194]
Accuracy: 0.821 ± 0.009
```

"""El OneHotEncoder generalmente funciona mejor con modelos lineales como la Regresión Logística. Esto se debe a que:  
- OrdinalEncoder asume un orden implícito entre las categorías ( $0 < 1 < 2 \dots$ )  
- Este orden arbitrario puede confundir a los modelos lineales  
- OneHotEncoder trata cada categoría de forma independiente sin asumir orden  
- Esto es más apropiado para variables categóricas nominales (sin orden natural)"""