Práctica 2

El objetivo de este ejercicio es evaluar el impacto del preprocesamiento de características en un **pipeline** que utiliza un clasificador basado en árboles de decisión en lugar de una regresión logística.

- La primera pregunta es evaluar empíricamente si escalar las características numéricas es útil o no;
- La segunda pregunta es evaluar si es empíricamente mejor (tanto desde una perspectiva computacional como estadística) utilizar categorías codificadas como enteros o codificadas con **one-hot**.

```
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"])
```

Al igual que en los notebooks anteriores, usamos la utilidad `make_column_selector` para seleccionar solo las columnas con un tipo de dato específico. Además, listamos de antemano todas las categorías para las columnas categóricas.

```
from sklearn.compose import make_column_selector as selector

numerical_columns_selector = selector(dtype_exclude=object)
categorical_columns_selector = selector(dtype_include=object)
numerical_columns = numerical_columns_selector(data)
categorical_columns = categorical_columns_selector(data)
```

Pipeline de referencia (sin escalado numérico y categorías codificadas como enteros)

Primero, vamos a medir el tiempo del pipeline que usamos en el notebook principal para que sirva como referencia:

```
import numpy as np

data_clean = data.copy()

data_clean = data_clean.replace('?', np.nan)

for col in categorical_columns:
    data_clean[col] = data_clean[col].fillna('Missing')

for col in numerical_columns:
    data_clean[col] = pd.to_numeric(data_clean[col], errors='coerce')
```

```
import time
from sklearn.model_selection import cross_validate
from sklearn.pipeline import make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.ensemble import HistGradientBoostingClassifier

categorical_preprocessor = OrdinalEncoder(
    handle_unknown="use_encoded_value", unknown_value=-1
)
preprocessor = ColumnTransformer(
    [("categorical", categorical_preprocessor, categorical_columns)],
    remainder="passthrough",
)

model = make_pipeline(preprocessor, HistGradientBoostingClassifier())

start = time.time()
cv_results = cross_validate(model, data_clean, target)
elapsed_time = time.time() - start

scores = cv_results["test_score"]
```

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

```
The mean cross-validation accuracy is: 0.812 ± 0.030
```

Escalado de características numéricas

Vamos a escribir una pipeline similar que también escale las características numéricas usando `StandardScaler` (o similar):

```
from sklearn.preprocessing import StandardScaler

categorical_preprocessor = OrdinalEncoder(
    handle_unknown="use_encoded_value", unknown_value=-1
)
numerical_preprocessor = StandardScaler()

preprocessor_with_scaling = ColumnTransformer(
    [
        ("categorical", categorical_preprocessor, categorical_columns),
        ("numerical", numerical_preprocessor, numerical_columns)
    ]
)

model_with_scaling = make_pipeline(
    preprocessor_with_scaling,
    HistGradientBoostingClassifier()
)

start = time.time()
cv_results_with_scaling = cross_validate(model_with_scaling, data, target)
elapsed_time_with_scaling = time.time() - start

scores_with_scaling = cv_results_with_scaling["test_score"]

print(
    "The mean cross-validation accuracy is: "
    f"{scores_with_scaling.mean():.3f} ± {scores_with_scaling.std():.3f} "
)
```

```
The mean cross-validation accuracy is: 0.810 ± 0.029
```

Vemos que escalar las variables numéricas no impacta en la precisión del modelo basado en árboles como sucedió en el modelo de regresión logística. En cuanto al tiempo de ejecución, vemos que los datos escalados aceleran un poco el entrenamiento en comparación con los datos no escalados del modelo de referencia anterior.

Codificación one-hot de variables categóricas

Observamos que la codificación de categorías como enteros puede ser muy perjudicial para los modelos lineales. Sin embargo, no parece ser el caso para los modelos `HistGradientBoostingClassifier`, ya que la puntuación de validación cruzada del pipeline de referencia con `OrdinalEncoder` es razonablemente buena.

Veamos si podemos obtener una precisión aún mejor con `OneHotEncoder` y escalando los datos numércos.

Pista: `HistGradientBoostingClassifier` aún no admite datos de entrada en formato sparse. Es posible que desees usar `OneHotEncoder(handle_unknown="ignore", sparse_output=False)` para forzar el uso de una representación densa como solución temporal.

```
from sklearn.preprocessing import OneHotEncoder, StandardScaler

categorical_preprocessor_onehot = OneHotEncoder(
    handle_unknown="ignore",
    sparse_output=False
)
numerical_preprocessor = StandardScaler()

preprocessor_onehot = ColumnTransformer(
    [
```

```
        ("onehot", categorical_preprocessor_onehot, categorical_columns),
        ("scaler", numerical_preprocessor, numerical_columns)
    ]
)

model_onehot = make_pipeline(
    preprocessor_onehot,
    HistGradientBoostingClassifier()
)

start = time.time()
cv_results_onehot = cross_validate(model_onehot, data_clean, target)
elapsed_time_onehot = time.time() - start

scores_onehot = cv_results_onehot["test_score"]

print(
    "The mean cross-validation accuracy is: "
    f"{scores_onehot.mean():.3f} ± {scores_onehot.std():.3f}"
)
```

```
The mean cross-validation accuracy is: 0.812 ± 0.030
```

Escribe tus conclusiones respondiendo a las siguientes preguntas:

En modelos no lineales como los basados en árboles...

1. ¿Vale la pena escalar los datos numéricos?, explica por qué.
2. ¿Cuál tipo de codificación (OrdinalEncoder o OneHotEncoder) recomendarías utilizar para los datos categóricos y por qué?
3. ¿Cuál tipo de codificación (OrdinalEncoder o OneHotEncoder) requiere de más tiempo para el entrenamiento?

```
"""1) No vale la pena escalar los datos numéricos en modelos basados en árboles.
Los modelos basados en árboles de decisión toman decisiones basándose en puntos
de corte (umbrales) en lugar de distancias entre valores."""
```

```
'1) No vale la pena escalar los datos numéricos en modelos basados en árboles.\nLos modelos basados en árboles de decisión toman d
ecisiones basándose en puntos \nde corte (umbrales) en lugar de distancias entre valores.'
```

```
"""2) OrdinalEncoder
Tiene mayor eficiencia computacional, menor uso de memoria y el tiempo de
entrenamiento es más rápido. Al final, el rendimiento es similar."""
```

```
'2) OrdinalEncoder\nTiene mayor eficiencia computacional, menor uso de memoria y el tiempo de \nentrenamiento es más rápido. Al fi
nal, el rendimiento es similar.'
```

```
"""3) OneHotEncoder
Tiene un aumento de dimensionalidad, ya que crea una nueva columna por cada
categoría única. Más decisiones por árbolporque cada nodo del árbol debe evaluar
más posibles divisiones. Y tiene mayor complejidad computacional, el algoritmo
debe procesar y almacenar más columnas durante el entrenamiento a comparación de OrdinalEncoder."""
```

```
'3) OneHotEncoder\nTiene un aumento de dimensionalidad, ya que crea una nueva columna por cada \ncategoría única. Más decisiones p
or árbolporque cada nodo del árbol debe evaluar\nmás posibles divisiones. Y tiene mayor complejidad computacional, el algoritmo \n
debe procesar y almacenar más columnas durante el entrenamiento a comparación de OrdinalEncoder.'
```