Trucos de preprocesamiento

Sesión 2 · Machine learning: regresión en Python · Andrés F. Puerta

1 · Cargar y mirar

```
pd.read_csv(ruta)  # lee un CSV, una URL o un .csv.gz
df.shape  # cuántas filas y columnas tiene
df.head(5)  # las primeras filas, para ver la pinta
df.info()  # tipos de cada columna y cuántos no-nulos
df.describe()  # resumen numérico: media, min, max, cuartiles
df.dtypes  # el tipo de cada columna
df["col"].value_counts()  # cuenta la frecuencia de cada valor
df["col"].value_counts(dropna=False)  # igual, contando también los vacíos
df["col"].unique()  # la lista de valores distintos
df["col"].nunique()  # cuántos valores distintos hay
df.select_dtypes(include="number")  # solo las columnas numéricas
df.select_dtypes(exclude="number")  # solo las de texto
df.drop(columns=["a", "b"])  # el df sin esas columnas
df.corr()  # correlación entre todas las numéricas
df[num].corrwith(y)  # correlación de cada una con el objetivo
```

2 · Valores faltantes

```
df.isna()  # True donde falta el dato
df.isna().sum()  # cuántos vacíos por columna
df.isna().mean() * 100  # porcentaje vacío de cada columna
df.dropna()  # fuera las filas con algún vacío
df.dropna(axis=1)  # fuera las columnas con algún vacío
df.dropna(axis=1, thresh=n)  # quita columnas con menos de n valores
df["col"].fillna(valor)  # rellena los vacíos con ese valor
df["col"].fillna(df["col"].median())  # los rellena con la mediana
df.groupby("g")["c"].transform("median")  # la mediana de c dentro de cada grupo
df["col"].isna().astype(int)  # columna 0/1: marca dónde faltaba
```

3 · Partir en entrenamiento y prueba

```
from sklearn.model_selection import train_test_split
train_test_split(X, y, test_size=0.2,  # parte los datos; random_state fija
                 random_state=42)  # el reparto para poder repetirlo
modelo.fit(X_train, y_train)  # APRENDE. Solo con entrenamiento
objeto.transform(X_test)  # APLICA lo aprendido. Nunca fit en test
objeto.fit_transform(X_train)  # las dos cosas de golpe, solo en train
```

4 · Imputar

```
from sklearn.impute import SimpleImputer
SimpleImputer(strategy="median")  # rellena con la mediana
SimpleImputer(strategy="mean")  # con la media
SimpleImputer(strategy="most_frequent")  # con la moda; vale para texto
SimpleImputer(strategy="constant",  # con un valor fijo: crea una
              fill_value="Sin dato")  # categoría propia para el vacío
SimpleImputer(add_indicator=True)  # añade una columna que marca el vacío
imputador.statistics_  # lo que aprendió para rellenar
from sklearn.impute import KNNImputer
KNNImputer(n_neighbors=5)  # rellena mirando a los k más parecidos
```

5 · Variables categóricas

```
(df["col"] == "valor").astype(int)  # binaria: convierte a 0 y 1
pd.get_dummies(df[["col"]], dtype=int)  # one-hot rápido, para explorar
from sklearn.preprocessing import OneHotEncoder
OneHotEncoder(handle_unknown="ignore")  # una categoría nueva sale como ceros
OneHotEncoder(drop="first")  # quita una: evita la colinealidad
OneHotEncoder(min_frequency=15)  # junta las categorías de menos de 15
encoder.get_feature_names_out()  # nombres de las columnas creadas
from sklearn.preprocessing import OrdinalEncoder
OrdinalEncoder(categories=[orden])  # ordinal CON el orden escrito a mano
df["col"].where(cond, "Otras")  # manda a "Otras" lo que no cumpla
df["col"].map(diccionario)  # sustituye valores con un diccionario
df["col"].astype(str)  # a texto: códigos que parecen números
```

6 · Agrupar y discretizar

```
df.groupby("col")["otra"].mean()  # la media de otra por grupo
df.groupby("col").agg(n=("x", "size"),  # varios resúmenes a la vez,
                      m=("x", "mean"))  # con el nombre que quieran
pd.cut(df["col"], bins=[0, 12, 20, 100])  # corta en tramos elegidos por ustedes
pd.qcut(df["col"], q=4)  # corta en grupos del mismo tamaño
```

7 · Escalar

```
from sklearn.preprocessing import StandardScaler
StandardScaler()  # media 0 y desviación 1. El defecto
MinMaxScaler()  # lo lleva todo al rango 0-1
RobustScaler()  # mediana y cuartiles: aguanta atípicos
escalador.mean_, escalador.scale_  # lo aprendido; hay que guardarlo
```

8 · Transformar el objetivo

```
np.log1p(y)  # endereza las colas a la derecha
np.expm1(pred)  # deshace el log1p
y.skew()  # asimetría; 0 es simétrico
from sklearn.compose import TransformedTargetRegressor
TransformedTargetRegressor(regressor=m,  # entrena con log(y) y devuelve
    func=np.log1p, inverse_func=np.expm1)  # las predicciones ya deshechas
```

9 · Pipelines

```
from sklearn.pipeline import Pipeline
Pipeline([("imp", SimpleImputer()),  # encadena pasos; el último es
          ("reg", LinearRegression())])  # el modelo
from sklearn.compose import ColumnTransformer
ColumnTransformer([("num", rama1, num_cols),  # aplica una rama distinta
                   ("cat", rama2, cat_cols)])  # a cada grupo de columnas
pipeline.named_steps["nombre"]  # accede a un paso concreto
pre.fit(X).transform(X).shape[1]  # cuántas columnas salen
```

10 · Medir

```
from sklearn.model_selection import cross_validate
cross_validate(t, X, y, cv=5,  # valida en 5 pliegues; devuelve un
    scoring=["r2", "neg_mean_squared_error"])  # dict con test_r2, etc.
cross_validate(..., error_score="raise")  # deja ver el error de verdad
from sklearn.metrics import mean_squared_error, r2_score
mean_squared_error(y, pred)  # MSE: error cuadrático medio
r2_score(y, pred)  # 1 perfecto, 0 como la media, <0 peor
DummyRegressor(strategy="mean")  # línea base: predice siempre la media
```

11 · Afinar

```
from sklearn.model_selection import GridSearchCV
GridSearchCV(pipeline, {"reg__alpha": [1, 10]},  # prueba combinaciones;
             cv=5, scoring="r2")  # doble guion bajo: paso__parámetro
cross_validate(GridSearchCV(...), X, y, cv=5)  # anidada: la búsqueda va DENTRO
busqueda.best_params_  # la combinación que ganó
```

La receta completa

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer, TransformedTargetRegressor
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import cross_validate

num_cols = X.select_dtypes(include="number").columns.tolist()
cat_cols = X.select_dtypes(exclude="number").columns.tolist()

rama_num = Pipeline([("imp", SimpleImputer(strategy="median")),
                     ("esc", StandardScaler())])

rama_cat = Pipeline([("imp", SimpleImputer(strategy="constant",
                                           fill_value="Desconocido")),
                     ("oh",  OneHotEncoder(handle_unknown="ignore",
                                           drop="first", min_frequency=15))])

pre = ColumnTransformer([("num", rama_num, num_cols),
                         ("cat", rama_cat, cat_cols)])

modelo = Pipeline([("pre", pre), ("reg", LinearRegression())])

r = cross_validate(modelo, X, y, cv=5, scoring=["r2", "neg_mean_squared_error"])
print(r["test_r2"].mean())
```

Qué usar según la columna

Numérica con vacíos	`SimpleImputer(median) + StandardScaler`
Numérica muy sesgada	`np.log1p, o PowerTransformer`
Categórica binaria	`(col == valor).astype(int)`
Categórica sin orden	`OneHotEncoder(drop='first', min_frequency=n)`
Categórica con orden	`OrdinalEncoder(categories=[orden escrito a mano])`
Categórica con 30+ niveles	`min_frequency, o agrupar por dominio`
NaN que significa 'no tiene'	`SimpleImputer(constant, fill_value='No_tiene')`
Número que en realidad es un código	`astype(str) y tratarla como nominal`
Objetivo sesgado	`TransformedTargetRegressor(log1p / expm1)`

El orden correcto

1. Cargar y MIRAR: shape, head, info, describe.
2. Quitar lo que no debería estar: identificadores y columnas que son la respuesta disfrazada.
3. Entender POR QUÉ falta lo que falta, antes de rellenarlo.
4. Partir en entrenamiento y prueba (o montar la validación cruzada).
5. Imputar, codificar y escalar SIEMPRE dentro de un Pipeline.
6. Medir contra una línea base y validar de forma cruzada.
7. Y solo entonces, cambiar de modelo y afinarlo.

Cinco avisos que valen más que cualquier truco

fit solo con entrenamiento

Medias, medianas, mínimos, categorías: todo lo que se aprenda de los datos se ajusta con train y se aplica a test. Un fit sobre todo el dataset es fuga, y no da ningún error.

Un NaN no siempre es un dato perdido

A veces significa 'no tiene'. Antes de imputar, pregúntense por qué falta. Si es una categoría, denle su propia categoría.

Label encoding no va en las variables de entrada

Le inventa un orden y unas distancias a lo que no las tiene. Para las nominales, one-hot; para las ordinales, OrdinalEncoder con el orden escrito a mano.

Miren la cardinalidad antes del one-hot

Una columna con muchos niveles genera muchas columnas casi vacías. Con más columnas que filas, el modelo memoriza.

No hay un preprocesamiento correcto en abstracto

Escalar no le hace nada a una regresión lineal y lo es todo para una red neuronal. Prueben con el modelo que van a usar.