Business Analytics & Data Science

Día 7: Evaluación de Modelos de Regresión

EAE Business School Barcelona

10 de febrero de 2026

Plan del Día 7

Primera Parte (9:00-11:00)

1. Repaso regresión simple
2. Métricas de evaluación: MAE, RMSE, R^2
3. Regresión lineal múltiple

Segunda Parte (11:30-13:30)

1. Feature engineering
2. One-hot encoding para categóricas
3. Overfitting y cómo detectarlo

Recap: Regresión Lineal Simple

Ayer vimos:

- ML workflow: datos → features → modelo → predicciones → evaluación
- Train/test split
- Regresión lineal OLS: $\hat{y} = \beta_0 + \beta_1 x$
- scikit-learn: `fit()`, `predict()`

Hoy: Evaluación y mejora de modelos

Primera Parte: Métricas de Evaluación

Objetivo

Cómo podemos decidir si nuestro modelo lo está haciendo bien?

Dicho de otra manera: entre dos modelos entrenados sobre el mismo dataset, cómo decidimos cuál es mejor?

Necesitamos métricas cuantitativas

Para regresión, medimos **error** = diferencia entre predicción y valor real

$$r = y - \hat{y}$$

Residuo = error en cada observación

Queremos: Errores pequeños en promedio

Dicho de otra manera: interesa minimizar $\overline{r}$.

MAE: Mean Absolute Error

MAE = promedio de errores absolutos

$$MAE = \frac{1}{n} \sum |y - \hat{y}|$$

```python
from sklearn.metrics import mean_absolute_error

mae = mean_absolute_error(y_test, y_pred)
print(f"MAE: {mae:.0f}€")
```

Ventaja: Fácil de interpretar, mismas unidades que y

MSE: Mean Squared Error

MSE = promedio de los cuadrados de los errores

$$MSE = \frac{1}{n}\sum(y - \hat{y})^2$$

```
from sklearn.metrics import mean_squared_error

rmse = mean_squared_error(y_test, y_pred)
print(f"MSE: {rmse:.0f}€")
```

Es otra medida absoluta de *error* del modelo. Dos modelos entrenados sobre el mismo dataset dan MSEs comparables entre si.

RMSE: Root Mean Squared Error

El MSE no es comparable con el MAE ya que el primero tiene unidades al cuadrado respecto del segundo. La solución es sacar una raíz cuadrada:

$$RMSE = \sqrt{MSE}$$

Interpretación:

Similar a MAE pero penaliza errores grandes. Siempre se cumple que

$$RMSE \geq MAE$$

ya que el primero castiga más a los outliers

R^2: Coeficiente de Determinación

R^2 es la proporción de varianza de la variable objetivo explicada por el modelo

$$R^2 = 1 - \frac{\sum(y - \hat{y})^2}{\sum(y - \overline{y})^2}$$

Razonamiento: un modelo es mejor cuanto mejor explica la variabilidad de la variable y.

Rango: 0 a 1 (puede ser negativo si modelo es peor que la media)

- $R^2 = 1 \rightarrow$ modelo perfecto
- $R^2 = 0.8 \rightarrow$ modelo explica 80% de varianza
- $R^2 \leq 0 \rightarrow$ modelo no mejor que predecir la media

Para calcular el R^2:

```
from sklearn.metrics import r2_score

r2 = r2_score(y_test, y_pred)
print(f"R²: {r2:.3f}")
```

Comparar Métricas

Métrica	Unidades	Interpretación	Deseable
MAE	Mismas que y	Error promedio	⬇️ Menor
RMSE	Mismas que y	Error con penalización outliers	⬇️ Menor
R^2	0 a 1	Varianza explicada	⬆️ Mayor

En práctica: Reportar las 3 métricas

Visualizar Predicciones

```
import plotly.express as px

df_results = pd.DataFrame({
    'Real': y_test,
    'Predicho': y_pred
})

fig = px.scatter(df_results, x='Real', y='Predicho',
                 title='Predicciones vs Reales')
fig.show()
```

Puntos cerca de la diagonal = buenas predicciones

Residual Plot

```
residuos = y_test - y_pred

fig = px.scatter(x=y_pred, y=residuos,
                 title='Residuos vs Predicciones',
                 labels={'x': 'Predicción', 'y': 'Residuo'})
fig.add_hline(y=0, line_dash='dash')
fig.show()
```

Buen modelo: Residuos distribuidos aleatoriamente alrededor de 0

Mal modelo: Patrón sistemático en residuos

Segunda Parte: Regresión Lineal Múltiple

Múltiples Features

Regresión simple: $precio = \beta_0 + \beta_1 \times metros$

Regresión múltiple: Usar MUCHAS features

$$precio = \beta_0 + \beta_1 \times metros + \beta_2 \times habitaciones + \beta_3 \times piso + ...$$

En Python - ¡exactamente igual!

```
X = df[['sqrmts', 'rooms', 'floor', 'bathrooms']]
y = df['price']

model = LinearRegression()
```

Interpretar Coeficientes Múltiples

```
print(f"Intercept: {model.intercept_:.0f}")
for feature, coef in zip(X.columns, model.coef_):
    print(f"{feature}: {coef:.0f}€")
```

Output ejemplo:

```
Intercept: 50000
sqrmts: 2500€
rooms: 15000€
floor: 3000€
```

🔬 Ahora al notebook

Ejercicio Evaluación y Regresión Múltiple

Tercera Parte: Feature Engineering

Feature Engineering

Arte de crear features útiles a partir de datos crudos

Ejemplos:

- `price_per_sqm = price / sqrmts`
- `total_rooms = rooms + bathrooms`
- `is_high_floor = floor > 5`
- `neighborhood_encoded` (one-hot)

Objetivo: Ayudar al modelo a aprender patrones

Variables Categóricas: One-Hot Encoding

Problema: Modelos solo entienden números, no texto

Solución: One-hot encoding

Ejemplo: `neighborhood` → columnas binarias

```
df_encoded = pd.get_dummies(df, columns=['neighborhood'], drop_first=True)
```

Antes:

nhood
Eixample
Gràcia

Después:

nhood_Eixample	nhood_Gràcia	neighborhood_Sants	...
1	0	0	...
0	1	0	...

Overfitting: El Enemigo

Overfitting = modelo "memoriza" datos de entrenamiento pero generaliza mal

Síntomas:

- Error bajo en train
- Error alto en test
- Diferencia grande entre train y test

Causas:

- Modelo muy complejo
- Muchas features
- Pocos datos de entrenamiento

Detectar Overfitting

Calculamos nuestras métricas de error en dos lugares:

- Training data: error en entrenamiento
- Test data: error con datos *previamente no vistos*

Prevenir Overfitting

1. **Más datos de entrenamiento**
2. **Menos features** (selección de features)
3. **Regularización** (penalizar complejidad)
4. **Cross-validation** (validación cruzada)

🔬 Ahora al notebook

Ejercicio Feature Engineering

Gracias!