Clase 071 — Regresión polinomial

Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 4 § Polynomial Regression.
Duración estimada: 50 min.

Clase 071 — Regresión polinomial

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 4 § Polynomial Regression. Duración estimada: 50 min.*

Objetivo

Que el alumno ajuste modelos lineales a relaciones no lineales usando PolynomialFeatures de scikit-learn, entienda la combinatoria de features que esto genera, y reconozca el riesgo de overfitting cuando el grado crece.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Transformar features con PolynomialFeatures(degree=d) y entender qué columnas produce.
2. Ajustar un LinearRegression sobre features polinómicas y graficar la curva resultante.
3. Calcular cuántas features genera grado d con n variables originales (combinatoria con repetición).
4. Diagnosticar overfitting comparando RMSE en train vs test al subir el grado.
5. Decidir cuándo usar interaction_only=True vs incluir potencias.

Temas

#	Tema	Por qué importa
1	Modelo lineal sobre features no lineales	Linealidad es en los coeficientes, no en l
2	PolynomialFeatures(degree, include_bias, i	API para expandir el feature space.
3	Combinatoria de features: C(n+d, d)	Explosión cuadrática/cúbica con n features
4	Overfitting con grado alto	Curva oscila para pasar por todos los punt
5	Validación con train/test split	Sin split, grado alto siempre "gana" en tr
6	Interaction-only vs full polynomial	Cuándo importan las potencias y cuándo sol

Definiciones y características

PolynomialFeatures

: Transformer de sklearn.preprocessing que genera todas las combinaciones polinómicas de las features de entrada hasta grado degree. Para x con grado 2: [1, x, x²]. Para [x, x] grado 2: [1, x, x, x², xx, x²].

degree

: Grado máximo del polinomio. Grado 1 no transforma (más bias). Grado 2–3 cubre la mayoría de curvaturas suaves. Grado ≥10 casi siempre es overfit.

interaction_only=True

: Genera solo productos cruzados entre features distintas, sin potencias (x², x² quedan fuera). Útil cuando el efecto no lineal viene de interacciones, no de curvatura individual.

include_bias=True (default)

: Agrega columna de 1's. Conviene ponerlo en False si después usás LinearRegression (que ya tiene fit_intercept=True) para no duplicar.

Explosión combinatoria

: La cantidad de features generadas es C(n+d, d) = (n+d)! / (n! d!). Con n=10 y d=5: 3003 features. Con n=100 y d=3: 176851. Escala feo.

Generalización

: Capacidad del modelo de funcionar en datos no vistos. Subir grado mejora train pero llega un punto donde test empeora — es el síntoma clásico de overfitting.

Modelo lineal sobre features no lineales

: Ajustar y = w + wx + wx² es resolver un problema lineal en (w, w, w) aunque la curva en x sea una parábola. Por eso LinearRegression basta tras transformar.

Dataset / recursos

Sintético: y = 0.5 x² + x + 2 + ruido_gaussiano, con x [-3, 3], n=100. Ideal para visualizar curva ajustada y comparar grados.

Ejercicios

1. Generar dataset cuadrático. x = np.linspace(-3, 3, 100) + ruido. Graficar scatter.

2. Ajustar grado 2. PolynomialFeatures(degree=2, include_bias=False) + LinearRegression. Imprimir coef_ e intercept_ y compararlos con los del DGP (0.5, 1, 2).

3. Grafo de curvas. Ajustar grados 1, 2, 5, 30 sobre el mismo dataset y plotear las 4 curvas superpuestas al scatter. Observar oscilaciones en grado 30.

4. Curva train/test vs grado. Para grado 1 a 20, calcular RMSE en train y en test. Graficar ambas curvas en función del grado. Identificar el punto donde test empieza a subir.

5. Combinatoria. Con n_features=3 de entrada, contar columnas que devuelve PolynomialFeatures(degree=4, include_bias=False). Verificar con la fórmula C(n+d, d) - 1.

Homework verificable

Notebook con: (a) dataset sintético y = sin(x) + ruido en x [0, 2π]; (b) ajustar polinomios de grado 1 a 15; (c) split 80/20; (d) graficar RMSE train vs test por grado; (e) reportar el grado óptimo según test; (f) graficar curva del grado óptimo encima del scatter.

Criterio de aceptación: El grado óptimo reportado está entre 3 y 7 (zona razonable para sin con ruido). Curva train decrece monótona; curva test tiene forma de U.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
LinearRegression da coef_ con 2 columnas d	Dejaste include_bias=True y LinearRegressi
RMSE train baja pero test explota al subir	Overfitting clásico. Fix: bajá el grado, o

MemoryError con grado 10 y 50 features	Combinatoria: C(60, 10) ≈ 75M. Fix: reducí
Coeficientes enormes (1e8) con grado alto	Síntoma de overfitting + colinealidad entr
Olvidaste transformar el test set	Aplicaste .fit_transform en train y .fit_t

Preguntas frecuentes

¿PolynomialFeatures es un modelo no lineal?

No. Es una transformación del input. El modelo (LinearRegression) sigue siendo lineal en los coeficientes. Lo que es no lineal es la relación entre x original e y.

¿Qué grado elegir por default?

Empezá con 2. Si el residual muestra patrón curvo, subí a 3. Más allá de 4–5 rara vez es necesario en problemas reales — si lo necesitás, probablemente quieras un modelo no lineal (árboles, kernel) en vez de subir grado.

¿Cuándo interaction_only=True?

Cuando sabés (o sospechás) que el efecto no lineal viene de combinaciones entre features (ej: precio × cantidad), no de curvatura individual de cada una. Reduce muchísimo la cantidad de columnas.

¿Por qué siempre escalar antes de PolynomialFeatures?

Porque x^2 y x^3 agrandan rangos: si x [0, 1000], entonces x^3 [0, 1e9]. Eso degrada el condicionamiento numérico y mata cualquier regularización posterior. Escalá primero con StandardScaler.

¿Cómo se relaciona esto con la clase 064?

Polinomial es el ejemplo canónico de modelo con alto bias en grado bajo y alta varianza en grado alto. La 064 formaliza ese trade-off con curvas de aprendizaje.

Referencias

- Géron, cap. 4 § Polynomial Regression y § Learning Curves.
- sklearn PolynomialFeatures
- sklearn user guide § Polynomial regression

Siguiente clase

Clase 072 — Curvas de aprendizaje y bias-variance tradeoff

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

Archivos complementarios

- notebook.ipynb