Clase 069 — Regresión lineal: ecuación normal vs gradient descent

Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 4 § Linear Regression.

Duración estimada: 60 min.

Clase 069 — Regresión lineal: ecuación normal vs gradient descent

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 4 § Linear Regression. Duración estimada: 60 min.*

Objetivo

Que el alumno entienda regresión lineal desde adentro: la hipótesis $\hat{y} = \theta^T x$, por qué se usa MSE como costo, las dos formas de resolverla (ecuación normal cerrada vs gradient descent iterativo), y cuándo conviene cada una según el tamaño del dataset y la cantidad de features.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Escribir la hipótesis lineal $\hat{y} = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n$ en forma matricial $X\theta$.
2. Derivar la ecuación normal $\theta = (X^T X)^{-1} X^T y$ y resolverla con NumPy.
3. Usar LinearRegression de sklearn y entender que internamente usa pseudoinversa SVD (más estable que la ecuación normal).
4. Comparar complejidad: ecuación normal $O(n^3)$ en features vs gradient descent $O(n)$ por iteración.
5. Justificar la elección entre forma cerrada y GD según n features y m muestras.

Temas

#	Tema	Por qué importa
1	Hipótesis lineal y notación vectorial	Base de todo modelo lineal (incluso logíst
2	MSE como función de costo	Convexa, derivable, mínimo global garantiz
3	Ecuación normal (forma cerrada)	Solución exacta en un solo paso.
4	Pseudoinversa SVD	Lo que sklearn usa por debajo — funciona a
5	Gradient descent (intuición)	Cuando n es grande, la ecuación normal n
6	Complejidad computacional	$O(n^3)$ vs $O(n)$ por iteración — define

Definiciones y características

Hipótesis lineal

: Predicción $\hat{y} = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n = \theta^T x$ (agregando $x_0 = 1$ para absorber el bias). En forma matricial sobre todo el dataset: $\hat{y} = X\theta$, donde X tiene shape $(m, n+1)$.

MSE (Mean Squared Error)

: Costo $\text{MSE}(\theta) = \frac{1}{m} \sum_{i=1}^{m} (\theta^T x^{(i)} - y^{(i)})^2$. Se elige porque es

convexa (un único mínimo global), diferenciable en todo el dominio y penaliza más errores grandes. RMSE es la raíz, en unidades del target.

Ecuación normal

: Solución cerrada que sale de igualar el gradiente del MSE a cero: $\hat{\theta} = (X^T X)^{-1} X^T y$. Da el óptimo exacto sin iteraciones. Requiere invertir una matriz $(n+1) \times (n+1)$.

Pseudoinversa de Moore-Penrose (X^+)

: Generalización de la inversa que siempre existe, incluso si $X^T X$ es singular (features colineales) o no cuadrada. Se computa vía SVD: $X = U \Sigma V^T \Rightarrow X^+ = V \Sigma^+ U^T$. Sklearn usa np.linalg.lstsq (basado en SVD) — más estable numéricamente que invertir $X^T X$.

Complejidad $O(n^3)$

: Invertir $X^T X$ cuesta entre $O(n^{2.4})$ y $O(n^3)$ según el algoritmo. Con $n = 100$ features va instantáneo; con $n = 100{,}000$ es inviable. La SVD es del mismo orden. En m (muestras) la ecuación normal es lineal, así que escala bien en filas, mal en columnas.

Gradient descent (intuición)

: Algoritmo iterativo: arrancás con θ random y vas restando $\eta \cdot \nabla_\theta \text{MSE}(\theta)$ hasta converger. Cada paso cuesta $O(mn)$ (batch) — mucho menos que invertir cuando n es grande. La tasa de aprendizaje η es el hiperparámetro crítico.

LinearRegression (sklearn)

: Estimador en sklearn.linear_model. No usa ecuación normal — usa scipy.linalg.lstsq (pseudoinversa SVD). Sin hiperparámetros relevantes salvo fit_intercept. Atributos post-fit: coef_ (los $\theta_1 \dots \theta_n$) e intercept_ (θ_0).

fit_intercept=True

: Agrega automáticamente la columna de unos. Si tus features ya están centradas en cero (mean=0), podés ponerlo en False. Default True — dejalo así salvo que sepas lo que hacés.

coef_

: Array con los pesos aprendidos, una entrada por feature. Su signo y magnitud son interpretables solo si las features están en la misma escala (de ahí la importancia de StandardScaler previo).

Dataset / recursos

Dataset sintético generado con NumPy: $y = 4 + 3x + \text{ruido gaussiano}$, $m = 100$ muestras. Permite comparar el θ recuperado con el verdadero $[4, 3]$.

Ejercicios

1. Hipótesis a mano. Generá X sintético ($m=100$, $n=1$) con $y = 4 + 3x + \mathcal{N}(0, 1)$. Resolvé $\hat{\theta}$ con la ecuación normal usando solo NumPy (np.linalg.inv, @). Verificá que $\hat{\theta} \approx [4, 3]$.

2. Pseudoinversa. Repetí el ejercicio 1 con np.linalg.pinv(X_b) @ y. Compará el resultado con la ecuación normal — deberían dar lo mismo en este caso bien-condicionado.

3. sklearn. Ajustá LinearRegression al mismo dataset. Verificá que lin_reg.intercept_ ≈ 4 y lin_reg.coef_ ≈ [3]. Predecí en $x = [[0], [2]]$.

4. Caso singular. Construí X con dos features colineales ($x_2 = 2 x_1$). Intentá la ecuación normal — np.linalg.inv tira LinAlgError o devuelve basura. Usá np.linalg.pinv y observá que sí funciona (distribuye el peso entre las dos features).

5. Complejidad empírica. Cronometrá LinearRegression().fit(X, y) con $n = 100, 1000, 10000$ features (y m fijo). Graficá el tiempo — debería crecer cúbicamente.

Homework verificable

Notebook con dataset California Housing (sklearn.datasets.fetch_california_housing): (a) split 80/20 con random_state=42; (b) ajustar LinearRegression; (c) reportar R^2 y RMSE en test; (d) imprimir coef_ mapeado a nombres de feature; (e) resolver manualmente la ecuación normal en los datos escalados y verificar que da los mismos coeficientes que sklearn (módulo escalado del intercept).

Criterio de aceptación: RMSE en test < 0.75 (en unidades de la variable target). Los coeficientes manuales coinciden con sklearn con tolerancia 1e-6.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
LinAlgError: Singular matrix al hacer np.l	Features colineales o $m < n$. Fix: usá np
coef_ enormes y de signo contrario a lo es	Features no escaladas y multicolinealidad.
Olvido de la columna de unos en la ecuació	Resolvés sin intercept y el modelo pasa po
LinearRegression da resultado distinto al	Olvidaste agregar la columna de unos en un
"Mi modelo lineal tarda horas con 50k feat	Ecuación normal $O(n^3)$ no escala. Fix: c

Preguntas frecuentes

¿Sklearn usa la ecuación normal por dentro?

No. Usa scipy.linalg.lstsq, que internamente factoriza con SVD (pseudoinversa). Es más estable numéricamente que $(X^T X)^{-1} X^T y$ y funciona aunque la matriz sea singular o rectangular.

¿Por qué MSE y no MAE como costo?

MSE es diferenciable en todo el dominio (MAE no lo es en cero) y convexa, así que el mínimo es único y se llega con gradient descent sin sustos. MAE es más robusto a outliers pero requiere métodos no-suaves (programación lineal o subgradiente).

¿Cuándo elijo ecuación normal vs gradient descent?

Regla práctica: n < ~10,000 features → ecuación normal/SVD (un shot, sin tunear hiperparámetros). n >> 10,000 o no entra en RAM → gradient descent (típicamente SGD). En m (muestras) ambos escalan bien, así que millones de filas con pocas columnas → ecuación normal sin drama.

¿Necesito escalar features para regresión lineal?

Para que el modelo funcione, no — la ecuación normal da exactamente la misma predicción con o sin escalado. Para interpretar coef_ (comparar importancia entre features), sí. Para gradient descent, sí o sí (sin escalar la convergencia es lentísima).

¿Qué pasa si tengo más features que muestras ($n > m$)?

$X^T X$ es singular → la ecuación normal falla. La pseudoinversa SVD devuelve la solución de norma mínima entre las infinitas posibles, pero el modelo va a sobreajustar feísimo. Fix real: regularización (Ridge/Lasso, clase 063) o reducción de dimensionalidad.

Referencias

- Géron, cap. 4 § Linear Regression, The Normal Equation, Computational Complexity.
- sklearn LinearRegression
- NumPy linalg.pinv
- Wikipedia — Moore-Penrose pseudoinverse

Siguiente clase

Clase 070 — Gradient Descent: batch, stochastic, mini-batch

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

Archivos complementarios

- notebook.ipynb