Geron

Outline breve (págs. 223 – 230)

- **Modelo de regresión lineal**
    - Predicción como suma ponderada de características + término de sesgo (Ecs. 4-1 y 4-2); notación vectorizada y^=θTx\hat{y}= \theta^{\mathsf T}x.
- **Función de coste**
    - Se adopta el error cuadrático medio (MSE) —o su raíz RMSE— para cuantificar el ajuste; se explicita la forma MSE(θ)\text{MSE}(\theta) (Ec. 4-3).
- **Entrenamiento por ecuación normal**
    - Solución cerrada θ^=(XTX)−1XTy\hat{\theta} = (X^{\mathsf T}X)^{-1}X^{\mathsf T}y (Ec. 4-4); demostración con NumPy: generación de datos, adición de columna de unos y cálculo de θ^\hat{\theta}.
- **Ejemplo NumPy + Matplotlib**
    - Datos sintéticos y=4+3x+ruidoy = 4 + 3x + \text{ruido}; ajuste produce θ0≈4.22\theta_0 \approx 4.22, θ1≈2.77\theta_1 \approx 2.77; gráfica del modelo frente a los puntos.
- **Alternativas computacionales**
    - `np.linalg.lstsq` (mínimos cuadrados vía SVD) y `np.linalg.pinv` (pseudoinversa de Moore-Penrose) evitan problemas si XTXX^{\mathsf T}X es singular; recomendados sobre invertir explícitamente.
- **Implementación en Scikit-Learn**
    - Uso de `LinearRegression`: llamada a `fit()`, atributos `intercept_` y `coef_`, y generación de predicciones con `predict()`.
- **Observaciones finales**
    - Ruido y tamaño de muestra condicionan la estimación de parámetros.

Regresión lineal

En el capítulo 1 analizamos un modelo de regresión simple de la satisfacción con la vida:

satisfacción_vida = $\theta_0 + \theta_1 \times$ PIB_per_cápita

Este modelo es simplemente una función lineal de la característica de entrada PIB per cápita. θ_0 y θ_1 son los parámetros del modelo.

De manera más general, un modelo lineal realiza una predicción simplemente calculando una suma ponderada de las características de entrada, más una constante llamada término de sesgo (también llamado término de intersección), como se muestra en la ecuación 4-1.

Ecuación 4-1. Predicción del modelo de regresión lineal

$$\hat{} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \quad + \theta_n x_n y$$

En esta ecuación:

- ŷ es el valor previsto.
- n es el número de características.
- x_i es el valor de la característica i .
- θ_j es el parámetro del modelo j , incluido el término de sesgo θ_0 y los pesos de las características $\theta_1, \theta_2, \ldots, \theta_n$.

Esto se puede escribir de forma mucho más concisa utilizando una forma vectorizada, como se muestra en la Ecuación 4-2.

Ecuación 4-2. Predicción del modelo de regresión lineal (forma vectorizada)

$$\hat{y} = h_\theta(x) = \theta \cdot x$$

En esta ecuación:

- h_θ es la función de hipótesis, utilizando los parámetros del modelo θ.
- θ es el vector de parámetros del modelo , que contiene el término de sesgo θ_0 y los pesos de las características θ_1 a θ_n

- x es el vector de características de la instancia, que contiene x a $_{norte}$, con x siempre x igual a 1.

- θ · x es el producto escalar de los vectores θ y x, que es igual a θ x + θx + θx + + θx 1 1 2.2 $_{nn}$.

> NOTA
>
> En el aprendizaje automático, los vectores a menudo se representan como vectores de columna, que son matrices 2D. con una sola columna. Si θ y x son vectores columna, entonces la predicción es y^=θ x, donde θ es la transpuesta de θ (un vector fila en lugar de un vector columna) y θ x es la matriz multiplicación de θ y x. Por supuesto, es la misma predicción, excepto que ahora es se representa como una matriz unicelular en lugar de un valor escalar. En este libro, usaré esto. Notación para evitar cambiar entre productos puntuales y multiplicaciones de matrices.

Bien, ese es el modelo de regresión lineal, pero ¿cómo lo entrenamos? Bueno, recuerden Que entrenar un modelo significa establecer sus parámetros para que el modelo se ajuste mejor el conjunto de entrenamiento. Para ello, primero necesitamos una medida de qué tan bien (o mal) el modelo se ajusta a los datos de entrenamiento. En el Capítulo 2 vimos que la mayoría La medida de rendimiento común de un modelo de regresión es la raíz cuadrada media. Error (Ecuación 2-1). Por lo tanto, para entrenar un modelo de regresión lineal, necesitamos Encuentre el valor de θ que minimiza el RMSE. En la práctica, es más sencillo minimizar el error cuadrático medio (MSE) que el RMSE, y conduce a la mismo resultado (porque el valor que minimiza una función positiva también minimiza su raíz cuadrada).

> ADVERTENCIA
>
> Los algoritmos de aprendizaje a menudo optimizarán una función de pérdida diferente durante el entrenamiento que la Medida de rendimiento utilizada para evaluar el modelo final. Esto se debe generalmente a que La función es más fácil de optimizar y/o porque tiene términos adicionales necesarios solo durante el entrenamiento (por ejemplo, para la regularización). Una buena métrica de rendimiento es la más cercana posible al resultado final. Objetivo empresarial. Una buena pérdida de entrenamiento es fácil de optimizar y está fuertemente correlacionada con la métrica. Por ejemplo, los clasificadores suelen entrenarse utilizando una función de coste como el logaritmo pérdida (como verá más adelante en este capítulo), pero evaluada mediante precisión/recuperación. La pérdida logarítmica es fácil de minimizar y hacerlo generalmente mejorará la precisión y la recuperación.

El MSE de una hipótesis de regresión lineal h en un conjunto de entrenamiento X se calcula utilizando la ecuación 4-3.

Ecuación 4-3. Función de costo MSE para un modelo de regresión lineal

$$\text{MSE}(X, h_\theta) = \frac{1}{m} \sum_{i=1}^{m} (\theta^{\top} x^{(i)} - y^{(i)})^2$$

La mayoría de estas notaciones se presentaron en el Capítulo 2 (véase “Notaciones”). La única diferencia es que escribimos h_θ en lugar de solo h para aclarar que el modelo está parametrizado por el vector θ. Para simplificar las notaciones, simplemente escribiremos MSE(θ) en lugar de MSE(X, h_θ).

La ecuación normal

Para hallar el valor de θ que minimiza el MSE, existe una solución cerrada; es decir, una ecuación matemática que proporciona el resultado directamente. Esta se denomina ecuación normal (Ecuación 4-4).

Ecuación 4-4. Ecuación normal

θ ^ = (X X) -1 X y

En esta ecuación:

- θ^ es el valor de θ que minimiza la función de costo.
- (m) y es el vector de valores objetivo que contienen y a (1) y .

Generemos algunos datos de aspecto lineal para probar esta ecuación (Figura 4-1):

```
importar numpy como np

np.random.seed(42) # para que este ejemplo de código sea reproducible
m = 100 # número de instancias X
= 2 * np.random.rand(m, 1) # vector de columna y = 4
+ 3 * X + np.random.randn(m, 1) # vector de columna
```

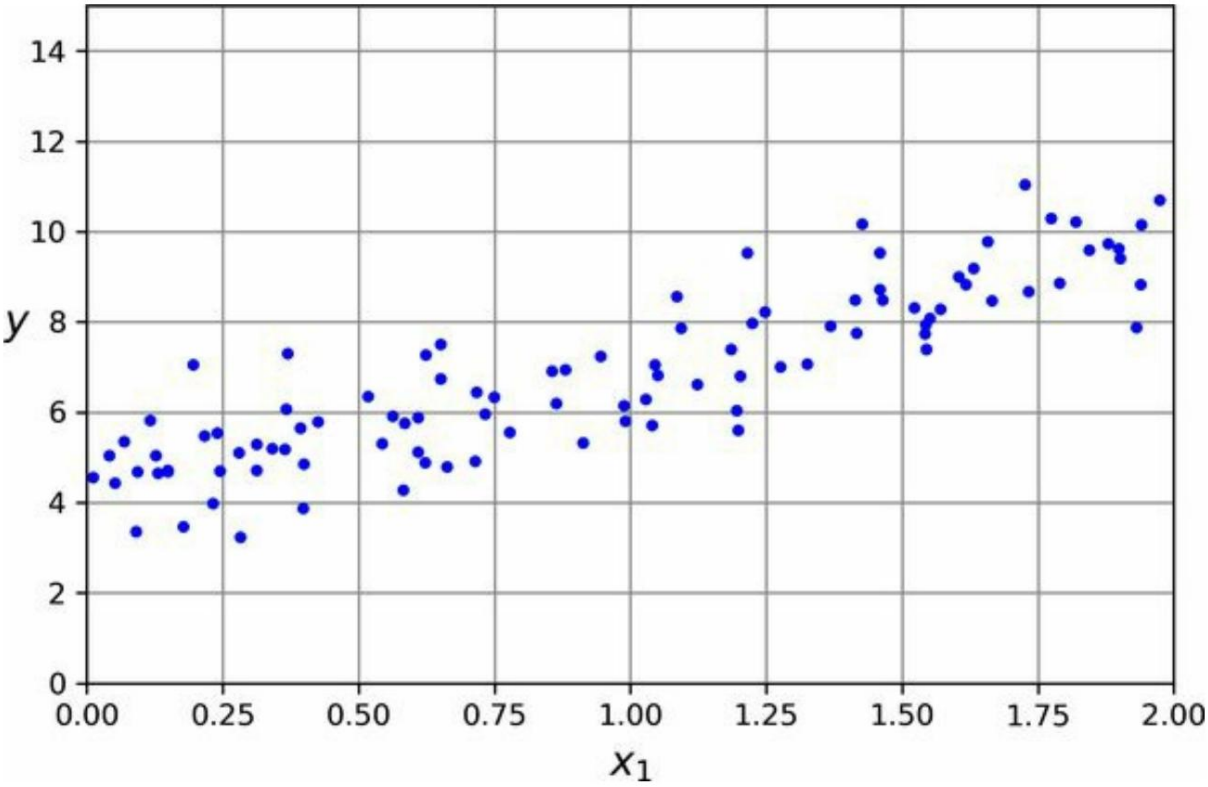

Figura 4-1. Un conjunto de datos lineales generados aleatoriamente

Ahora calculemos θ^ usando la ecuación normal. Usaremos la función inv() del módulo de álgebra lineal de NumPy (np.linalg) para calcular la inversa de una matriz y el método dot() para la multiplicación de matrices:

```
desde sklearn.preprocessing importar add_dummy_feature

X_b = add_dummy_feature(X) # agrega x0 = 1 a cada instancia
theta_best = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y
```

> NOTA
>
> El operador @ realiza la multiplicación de matrices. Si A y B son arrays de NumPy, entonces A @ B equivale a np.matmul(A, B). Muchas otras bibliotecas, como TensorFlow, PyTorch y JAX, también admiten el operador @. Sin embargo, no se puede usar @ en arrays de Python puro (es decir, listas de listas).

La función que usamos para generar los datos es y = 4 + 3x ₊ ruido gaussiano. Veamos el resultado de la ecuación:

```
>>> theta_best
matriz([[4.21509616],
        [2.77011339]])
```

Habríamos esperado $\theta_0 = 4$ y $\theta_1 = 3$ en lugar de $\theta_0 = 4,215$ y $\theta_1 = 2,770$. Bastante cerca, pero el ruido impidió recuperar los parámetros exactos de la función original. Cuanto más pequeño y ruidoso sea el conjunto de datos, más difícil se vuelve.

Ahora podemos hacer predicciones usando $\hat{\theta}$:

```
>>> X_nuevo = np.array([[0], [2]])
>>> X_new_b = add_dummy_feature(X_new) # agrega x0 = 1 a cada instancia
>>> y_predict = X_new_b @ theta_best
>>> y_predict
array([[4.21509616],
        [9.75532293]])
```

Grafiquemos las predicciones de este modelo (Figura 4-2):

```
importar matplotlib.pyplot como plt

plt.plot(X_new, y_predict, "r-", label="Predictions")
plt.plot(X, y, "b.")
[...] # embellecer la figura: agregar etiquetas, eje, cuadrícula y
leyenda plt.show()
```

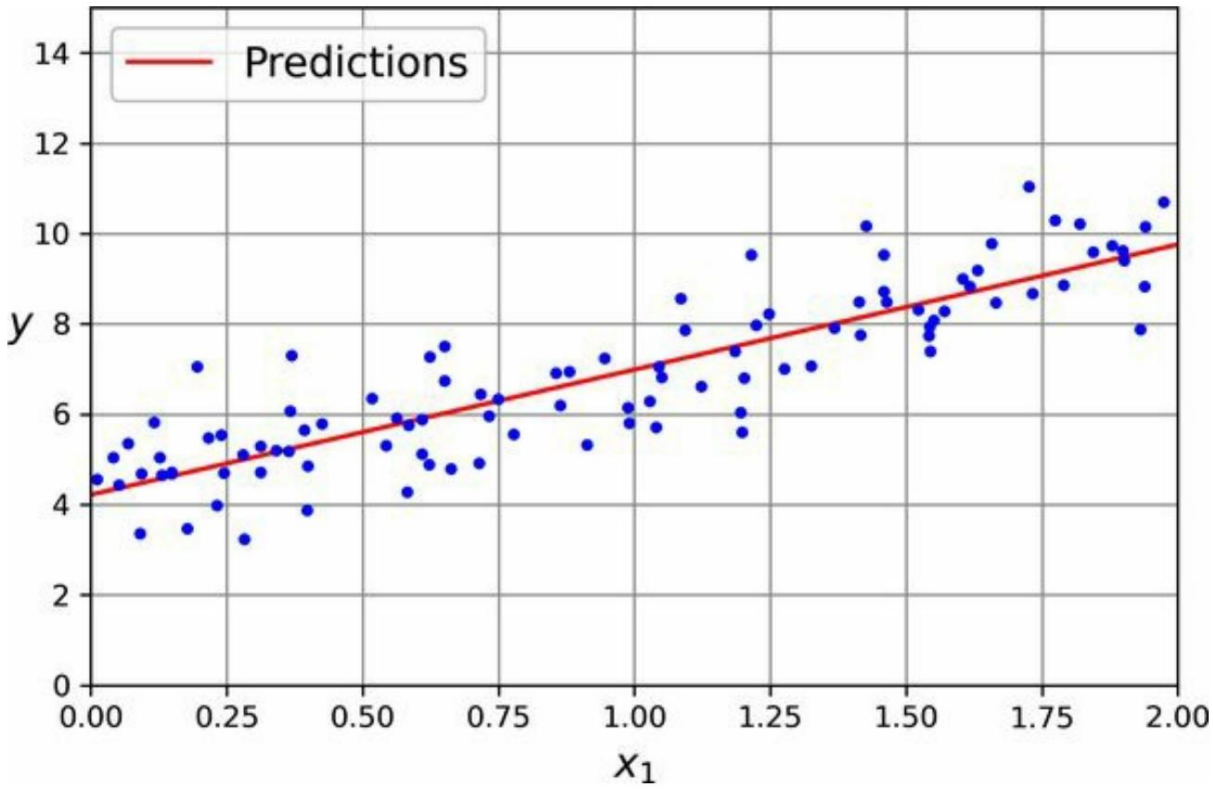

Figura 4-2. Predicciones del modelo de regresión lineal

Realizar una regresión lineal utilizando Scikit-Learn es relativamente sencillo:

```
>>> de sklearn.linear_model importar Regresión lineal >>> lin_reg
= Regresión lineal() >>> lin_reg.fit(X,
y) >>> lin_reg.intercept_,
lin_reg.coef_ (matriz([4.21509616]),
matriz([[2.77011339]])) >>> lin_reg.predict(X_new)
matriz([[4.21509616],
[9.75532293]])
```

Tenga en cuenta que Scikit-Learn separa el término de sesgo (intercept_) de las ponderaciones de las características (coef_). La clase LinearRegression se basa en la función scipy.linalg.lstsq() (el nombre significa "mínimos cuadrados"), que puede llamar directamente:

```
>>> theta_best_svd, residuos, rango, s = np.linalg.lstsq(X_b, y, rcond=1e-6) >>>
theta_best_svd
array([[4.21509616],
       [2.77011339]])
```

Esta función calcula θ^=X+y, donde X+ es la pseudoinversa de X (específicamente, la inversa de Moore-Penrose). Puede usar np.linalg.pinv() para calcular la pseudoinversa directamente:

```
>>> np.linalg.pinv(X_b) @ y
matriz([[4.21509616],
       [2.77011339]])
```

La pseudoinversa se calcula mediante una técnica estándar de factorización matricial denominada descomposición en valores singulares (SVD), que permite descomponer la matriz X del conjunto de entrenamiento en la multiplicación matricial de tres matrices U Σ V (véase numpy.linalg.svd()). La pseudoinversa se calcula como X+=VΣ+U . Para calcular la matriz Σ+, el algoritmo toma Σ y establece en cero todos los valores menores que un valor umbral minúsculo, luego reemplaza todos los valores distintos de cero con su inversa y, finalmente, transpone la matriz resultante. Este enfoque es más eficiente que calcular la ecuación normal, además de que gestiona bien los casos extremos: de hecho, la ecuación normal puede no funcionar si la matriz XX no es invertible (es decir, singular), como si m < n o si algunas características son redundantes, pero la pseudoinversa siempre está definida.