

Régressão Linear

Slides extraídos do livro “*Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*” de Aurélien Géron

Sumário

- Modelo
- Treinamento
- Equação Normal
- Exemplo
- Complexidade

Até agora, tratamos os modelos de AM como caixas-pretas. Neste capítulo, vamos abri-las.

- Ter um bom entendimento de como as coisas funcionam pode ajudá-lo a:
 - Escolher o modelo apropriado.
 - Escolher o algoritmo de treinamento certo.
 - Encontrar um bom conjunto de hiperparâmetros.
 - Depurar problemas e realizar análise de erros de forma eficiente.
- Começaremos com o modelo de **Regressão Linear**, um dos mais simples.
Vamos discutir duas **maneiras** muito diferentes de **treiná-lo**:
 - Usando uma **equação de "forma fechada"** que calcula diretamente os parâmetros do modelo.
 - Usando uma abordagem de otimização iterativa chamada **Gradiente Descendente (GD)**.

Regressão Linear - O Modelo

Um modelo linear faz uma previsão calculando uma soma ponderada das features de entrada, mais uma constante chamada de termo de viés (bias term).

Equação 4-1. Previsão do modelo de Regressão Linear

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

- \hat{y} é o valor previsto.
- n é o número de features.
- x_i é o valor da i-ésima feature.
- θ_j é o j-ésimo parâmetro do modelo (incluindo o termo de viés θ_0 e os pesos das features $\theta_1, \dots, \theta_n$).

Pode ser escrita de forma mais concisa usando uma forma vetorializada:

Equação 4-2. Previsão do modelo de Regressão Linear (forma vetorializada)

$$\hat{y} = h_{\theta}(x) = \theta^T \cdot x, \quad \text{onde } \theta = [\theta_0, \theta_1, \dots, \theta_n] \text{ e } x = [1, x_1, \dots, x_n].$$

- Treinar um modelo significa ajustar seus parâmetros para que o modelo se ajuste melhor ao conjunto de treinamento.
- Para isso, precisamos de uma medida de quanto bem (ou mal) o modelo se ajusta aos dados.
- A medida de desempenho mais comum para um modelo de regressão é a Raiz do Erro Quadrático Médio (RMSE).
- Na prática, é mais simples minimizar o Erro Quadrático Médio (MSE), e isso leva ao mesmo resultado.

Equação 4-3. Função de custo MSE para um modelo de Regressão Linear

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

Para encontrar o valor de θ que minimiza a função de custo, existe uma solução de forma fechada — em outras palavras, uma equação matemática que dá o resultado diretamente.

Equação 4-4. Equação Normal

$$\hat{\theta} = (X^T \cdot X)^{-1} \cdot X^T \cdot y$$

- $\hat{\theta}$ é o valor de θ que minimiza a função de custo.
- y é o vetor de valores alvo contendo $y^{(1)}$ até $y^{(m)}$.

Vamos gerar alguns dados com aparência linear para testar esta equação.

Exemplo Prático: Geração de Dados

```
import numpy as np

np.random.seed(42)
m = 100 # number of instances
X = 2 * np.random.rand(m, 1) # column vector
y = 4 + 3 * X + np.random.randn(m, 1) # column vector
```

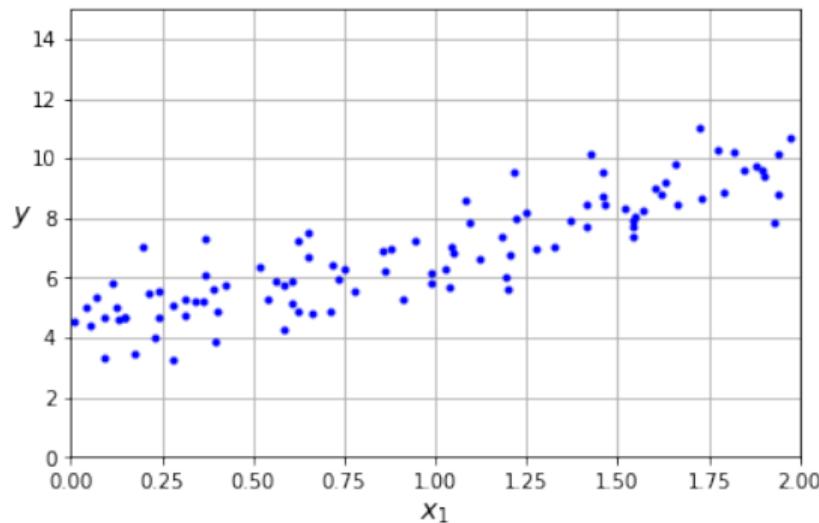


Figure: Dataset linear gerado aleatoriamente.

Exemplo Prático: Cálculo com a Equação Normal

Vamos calcular $\hat{\theta}$ usando a Equação Normal. Usaremos a função `inv()` do módulo de álgebra linear do NumPy e o método `dot()` para multiplicação de matrizes.

```
from sklearn.preprocessing import add_dummy_feature  
  
X_b = add_dummy_feature(X) # add x0 = 1 to each instance  
theta_best = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y
```

A função que usamos para gerar os dados foi $y = 4 + 3x_1 + \text{ruído Gaussiano}$.

Vamos ver o que a equação encontrou:

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

Esperávamos $\theta_0 = 4$ e $\theta_1 = 3$.

Próximo o suficiente, mas o ruído tornou impossível recuperar os parâmetros exatos.

Exemplo Prático: Previsões

Agora podemos fazer previsões usando $\hat{\theta}$:

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

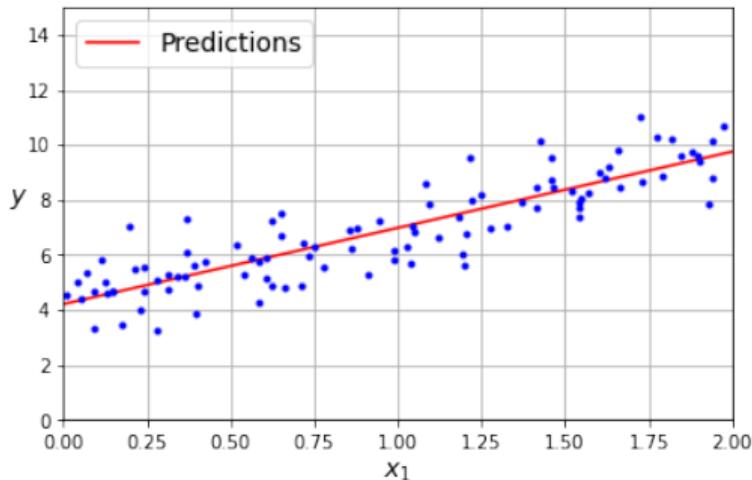


Figure: Previsões do modelo de Regressão Linear.

Regressão Linear com Scikit-Learn

Realizar Regressão Linear usando Scikit-Learn é relativamente simples:

```
>>> from sklearn.linear_model import LinearRegression  
>>> lin_reg = LinearRegression()  
>>> lin_reg.fit(X, y)  
>>> lin_reg.intercept_, lin_reg.coef_  
(array([4.21509616]), array([[2.77011339]]))  
>>> lin_reg.predict(X_new)  
array([[4.21509616],  
       [9.75532293]])
```

- A classe `LinearRegression` é baseada na função `scipy.linalg.lstsq()` (`least squares`).
- Esta função calcula $\hat{\theta} = X^+y$, onde X^+ é a **pseudoinversa** de X .
- A pseudoinversa é calculada usando uma técnica de fatoração de matriz padrão chamada **Decomposição em Valores Singulares (SVD)**.
- Esta abordagem é mais eficiente que a Equação Normal e lida bem com casos extremos.

- A **Equação Normal** calcula a inversa de $X^T \cdot X$, que é uma matriz $(n + 1) \times (n + 1)$. A complexidade computacional de inverter tal matriz é tipicamente cerca de $\mathcal{O}(n^{2.4})$ a $\mathcal{O}(n^3)$.
- A abordagem **SVD** usada pela classe `LinearRegression` do Scikit-Learn é cerca de $\mathcal{O}(n^2)$.
- **Alerta:** Ambas as abordagens se tornam muito lentas quando o número de features (n) cresce muito (ex: 100.000).
- No lado positivo, ambas são lineares em relação ao número de instâncias no conjunto de treinamento ($\mathcal{O}(m)$), então elas lidam com grandes conjuntos de treinamento eficientemente, desde que caibam na memória.
- As **previsões** são muito rápidas: a complexidade é linear em relação ao número de instâncias e de features.

Fim