Gradiente Descendente

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

Sumário

Sumário

Gradiente Descendente (GD)

O Gradiente Descendente é um algoritmo de otimização genérico capaz de encontrar soluções ótimas para uma ampla gama de problemas.

Ideia Geral

Ajustar os parâmetros iterativamente para minimizar uma função de custo.

- Ele mede o gradiente local da função de erro em relação ao vetor de parâmetros θ e vai na direção do gradiente descendente.
- Um parâmetro importante é o tamanho dos passos, determinado pelo hiperparâmetro **taxa de aprendizagem** (learning rate).

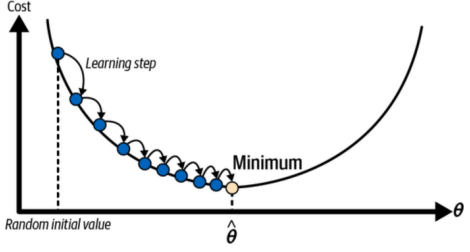
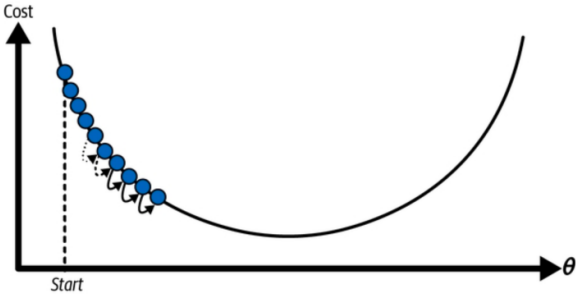


Figure: Os parâmetros são inicializados aleatoriamente e ajustados repetidamente para minimizar a função de custo.

A Taxa de Aprendizagem

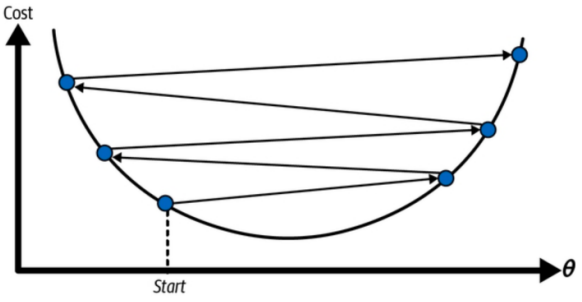
Taxa de Aprendizagem Pequena

- O algoritmo terá que passar por muitas iterações para convergir, o que levará muito tempo.



Taxa de Aprendizagem Grande

- Você pode pular o vale e acabar do outro lado, possivelmente mais alto do que antes. Isso pode fazer o algoritmo divergir.



Desafios do GD

Nem todas as funções de custo parecem tigelas regulares. Pode haver buracos, cumes, platôs e terrenos irregulares, dificultando a convergência.

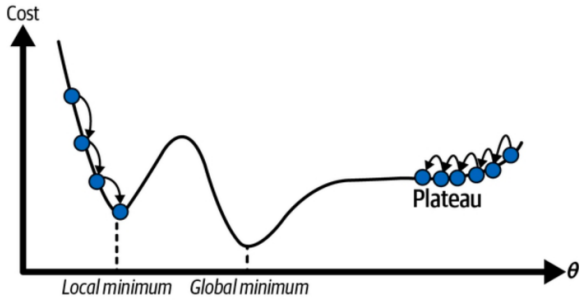


Figure: Se a inicialização aleatória começar à esquerda, o algoritmo convergirá para um mínimo local. Se começar à direita, levará muito tempo para cruzar o platô.

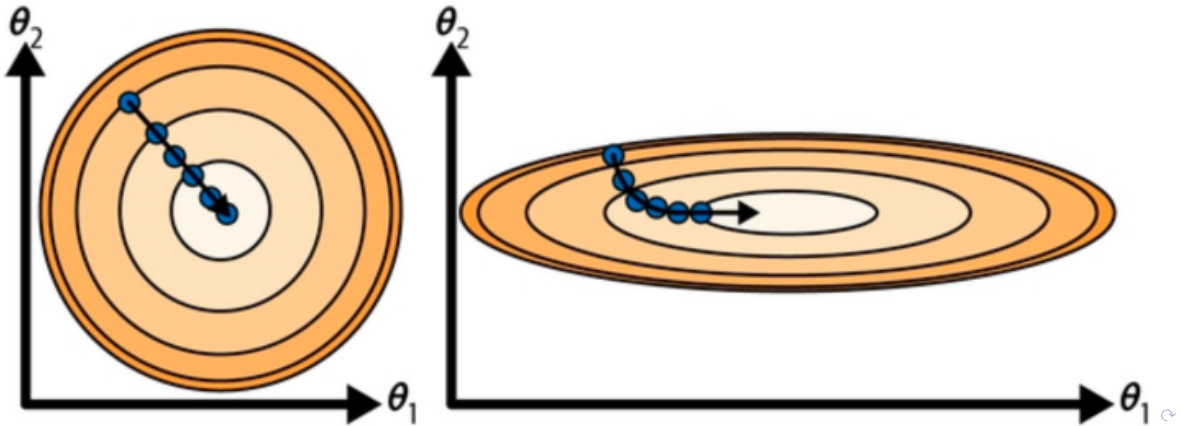
Boa Notícia

A função de custo MSE para um modelo de Regressão Linear é uma **função convexa**, o que significa que não há mínimos locais, apenas um mínimo global. O GD tem garantia de se aproximar arbitrariamente do mínimo global.

Importância da Escala das Features

Aviso

Ao usar o Gradiente Descendente, você deve garantir que todas as features tenham uma escala semelhante (por exemplo, usando a classe `StandardScaler` do Scikit-Learn), caso contrário, levará muito mais tempo para convergir.



Sumário

Gradiente Descendente em Lote (Batch GD)

Para implementar o GD, você precisa calcular o gradiente da função de custo em relação a cada parâmetro do modelo θ_j. Isso é chamado de **derivada parcial**.

Equação 4-5. Derivadas parciais da função de custo (MSE da Regressão Linear)

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\theta) = \frac{2}{m} \sum_{i=1}^{m} (\theta^T \cdot x^{(i)} - y^{(i)}) x_j^{(i)}$$

Em vez de calcular essas derivadas parciais individualmente, você pode usar a Equação 4-6 para calculá-las todas de uma vez. O vetor gradiente, notado $\nabla_\theta \text{MSE}(\theta)$, contém todas as derivadas parciais da função de custo.

Equação 4-6. Vetor gradiente da função de custo

$$\nabla_\theta \text{MSE}(\theta) = \frac{2}{m} X^T (X\theta - y)$$

Aviso

Esta fórmula envolve cálculos sobre todo o conjunto de treinamento X a cada passo do GD! É por isso que o algoritmo é chamado de Gradiente Descendente em Lote. Como resultado, é terrivelmente lento em conjuntos de treinamento muito grandes.

Passo do Gradiente Descendente

Uma vez que você tem o vetor gradiente, que aponta para cima, basta ir na direção oposta para descer. Isso significa subtrair $\nabla_\theta \text{MSE}(\theta)$ de θ.

É aqui que a taxa de aprendizagem η aparece: multiplique o vetor gradiente por η para determinar o tamanho do passo descendente.

Equação 4-7. Passo do Gradiente Descendente

$$\theta^{(\text{próximo passo})} = \theta - \eta \cdot \nabla_\theta \text{MSE}(\theta)$$

Implementação do Batch GD

Neste exemplo, o conjunto de treino é o mesmo usado no exemplo da Regressão Linear.

```
eta = 0.1  # learning rate
n_epochs = 1000
m = len(X_b)  # number of instances

np.random.seed(42)
theta = np.random.randn(2, 1)  # randomly initialized

for epoch in range(n_epochs):
    gradients = 2 / m * X_b.T @ (X_b @ theta - y)
    theta = theta - eta * gradients
```

Cada iteração sobre o conjunto de treinamento é chamada de **época**. Vamos ver o `theta` resultante:

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

É exatamente o que a Equação Normal encontrou! O Gradiente Descendente funcionou perfeitamente.

Efeito da Taxa de Aprendizagem no Batch GD

A figura mostra os primeiros 20 passos do Gradiente Descendente usando três taxas de aprendizagem diferentes.

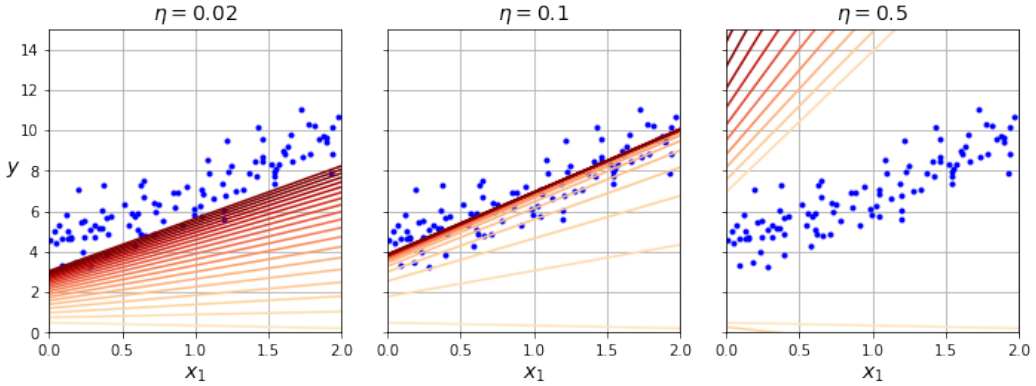


Figure: Gradiente Descendente com várias taxas de aprendizagem.

- **Esquerda**: A taxa de aprendizagem é muito baixa. Chegará à solução, mas levará muito tempo.
- **Meio**: Taxa de aprendizagem parece boa. Em poucas épocas, convergiu p/ solução.
- **Direita**: A taxa de aprendizagem é muito alta. O algoritmo diverge, saltando por todo o lugar.

Sumário

Gradiente Descendente Estocástico (Stochastic GD)

- O principal problema com o Batch GD é que ele usa todo o conjunto de treinamento para calcular os gradientes a cada passo.
- O **Stochastic GD** escolhe uma instância aleatória no conjunto de treinamento a cada passo e calcula os gradientes com base apenas nessa única instância.

Vantagens:

- Muito mais rápido.
- Permite treinar em conjuntos de dados enormes (out-of-core).
- A aleatoriedade ajuda a escapar de mínimos locais.

Desvantagens:

- Muito menos regular que o Batch GD; a função de custo salta para cima e para baixo.
- Nunca se estabelece no mínimo, continua a saltar ao redor.

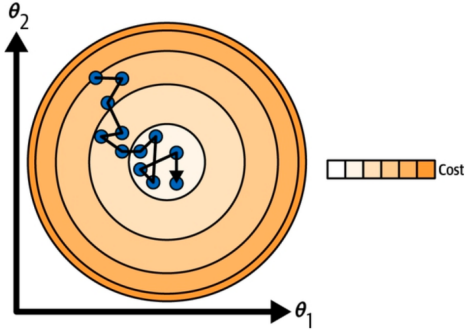


Figure: Com o Stochastic GD, cada passo de treinamento é muito mais rápido, mas também muito mais estocástico.

Solução para a instabilidade

Reduzir gradualmente a taxa de aprendizagem. A função que determina a taxa de aprendizagem em cada iteração é chamada de **agendamento de aprendizagem** (learning schedule).

Implementação do Stochastic GD

```
n_epochs = 50
t0, t1 = 5, 50  # learning schedule hyperparameters

def learning_schedule(t):
    return t0 / (t + t1)

np.random.seed(42)
theta = np.random.randn(2, 1)  # random initialization

for epoch in range(n_epochs):
    for iteration in range(m):
        random_index = np.random.randint(m)
        xi = X_b[random_index : random_index + 1]
        yi = y[random_index : random_index + 1]
        gradients = 2 * xi.T @ (xi @ theta - yi)
        eta = learning_schedule(epoch * m + iteration) # t aumenta 1 p/ chamada: eta ~ 0.1 até 0.001
        theta = theta - eta * gradients
```

Obs.: Indexar array Numpy por range preserva o número de dimensões.

Implementação do Stochastic GD

```
>>> theta
array([[4.21076011],
       [2.74856079]])
```

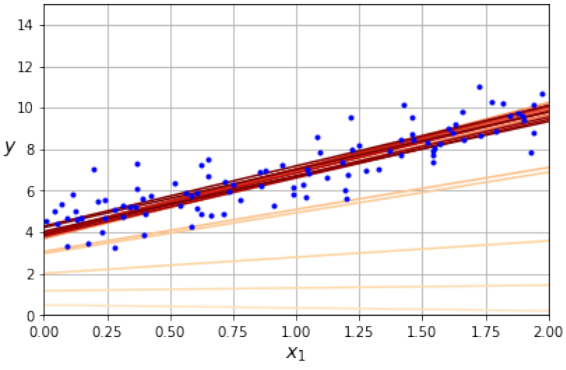


Figure: Os primeiros 20 passos do Stochastic GD.

Stochastic GD com Scikit-Learn

Para realizar Regressão Linear usando Stochastic GD com Scikit-Learn, você pode usar a classe `SGDRegressor`.

```
from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor(max_iter=1000, tol=1e-5, n_iter_no_change=100,
                       eta0=0.01, penalty=None, random_state=42)
sgd_reg.fit(X, y)
```

O código roda por no máximo 1000 épocas ou até que o custo caia menos de 10^{-5} durante 100 épocas. Penalty é o tipo de regularização (None para nenhuma regularização).

```
>>> sgd_reg.intercept_, sgd_reg.coef_
(array([4.21278812]), array([2.77270267]))
```

A solução é bastante próxima daquela retornada pela Equação Normal.

Sumário

Gradiente Descendente Mini-batch (Mini-batch GD)

- A cada passo, em vez de calcular os gradientes com base no conjunto de treinamento completo (como no Batch GD) ou com base em apenas uma instância (como no Stochastic GD), o Mini-batch GD calcula os gradientes em pequenos conjuntos aleatórios de instâncias chamados **mini-batches**.
- **Vantagem principal sobre o SGD**: Você pode obter um aumento de desempenho com a otimização de hardware de operações de matriz, especialmente ao usar GPUs.
- O progresso do algoritmo no espaço de parâmetros é menos errático que o do SGD, especialmente com mini-batches razoavelmente grandes.

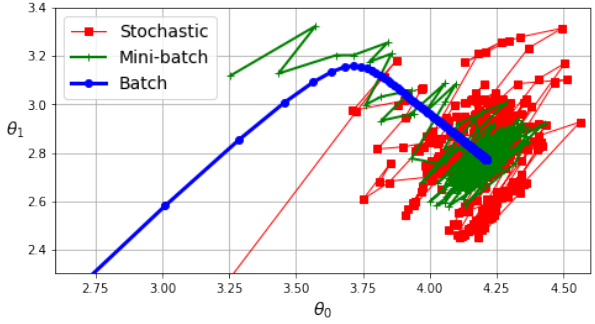


Figure: Caminhos do Gradiente Descendente no espaço de parâmetros.

No Scikit-Learn, o método `partial_fit` da classe `SGDRegressor` permite fornecer o conjunto de treino e rótulos para uma iteração do GD.

Assim, você pode implementar o Mini-batch GD manualmente.

Comparação dos Algoritmos para Regressão Linear

Table: Comparação de algoritmos para Regressão Linear

Algoritmo	m grande	Suporte Out-of-core	n grande	Hiperparams	Scaling required
Equação Normal	Rápido	Não	Lento	0	Não
SVD	Rápido	Não	Lento	0	Não
Batch GD	Lento	Não	Rápido	2	Sim
Stochastic GD	Rápido	Sim	Rápido	≥ 2	Sim
Mini-batch GD	Rápido	Sim	Rápido	≥ 2	Sim

m: número de instâncias de treinamento, n: número de features.

Conclusão

Após o treinamento, quase não há diferença: todos esses algoritmos acabam com modelos muito semelhantes e fazem previsões exatamente da mesma maneira.

Fim