Modelagem Analítica com Machine Learning

Segundo Dia

Paulo Cysne Rios, Jr.

Regressão Linear com mais de uma variável independente

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

De forma matricial

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

Diferença

A diferença entre o
valor real e o valor que predizemos

A diferença entre o valor predito e o valor real
pode ser expresso através do mean square error (MSE)

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

Valor Predito

Valor Real

Nosso objetivo é minimizar este valor!!

Minimizar esta diferença = predizer melhor!!

Minimizando o custo

Esta diferença é a nossa função custo

Podemos matematicamente minimizar esta função custo
Escolhendo coeficientes que a minimizem

Matematicamente se pode provar que estes valores dos coeficientes minimizam esta função custo

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

Consequências da Equação de Minimização do Custo

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

- Esta equação depende do produto de matrizes X.

- Sua computação se torna bastante lenta quando o número de variáveis independentes se torna muito grande (por exemplo, acima de 100 mil como no caso de aplicações em biologia molecular).

Consequências da Equação de Minimização do Custo

- Mas esta equação é linear em relação ao número de observações/linhas.

- Quer dizer, ela pode lidar com conjuntos de dados bastante grandes (muitas instâncias/observações).

- Conquanto que estes encontrem lugar na memória do computador (assumindo não se ter nenhuma estrutura de processamento distribuído como Hadoop/Stark).

Outra solução: Gradient Descent

- O que fazer quando se tem muitas variáveis independentes

- Ou não muito espaço na memória?

- Se pode usar Gradient Descent.

- Que não tem estas limitações!

- Uma solução usada em outras modelagens, inclusive em Deep Learning que veremos no próximo curso.

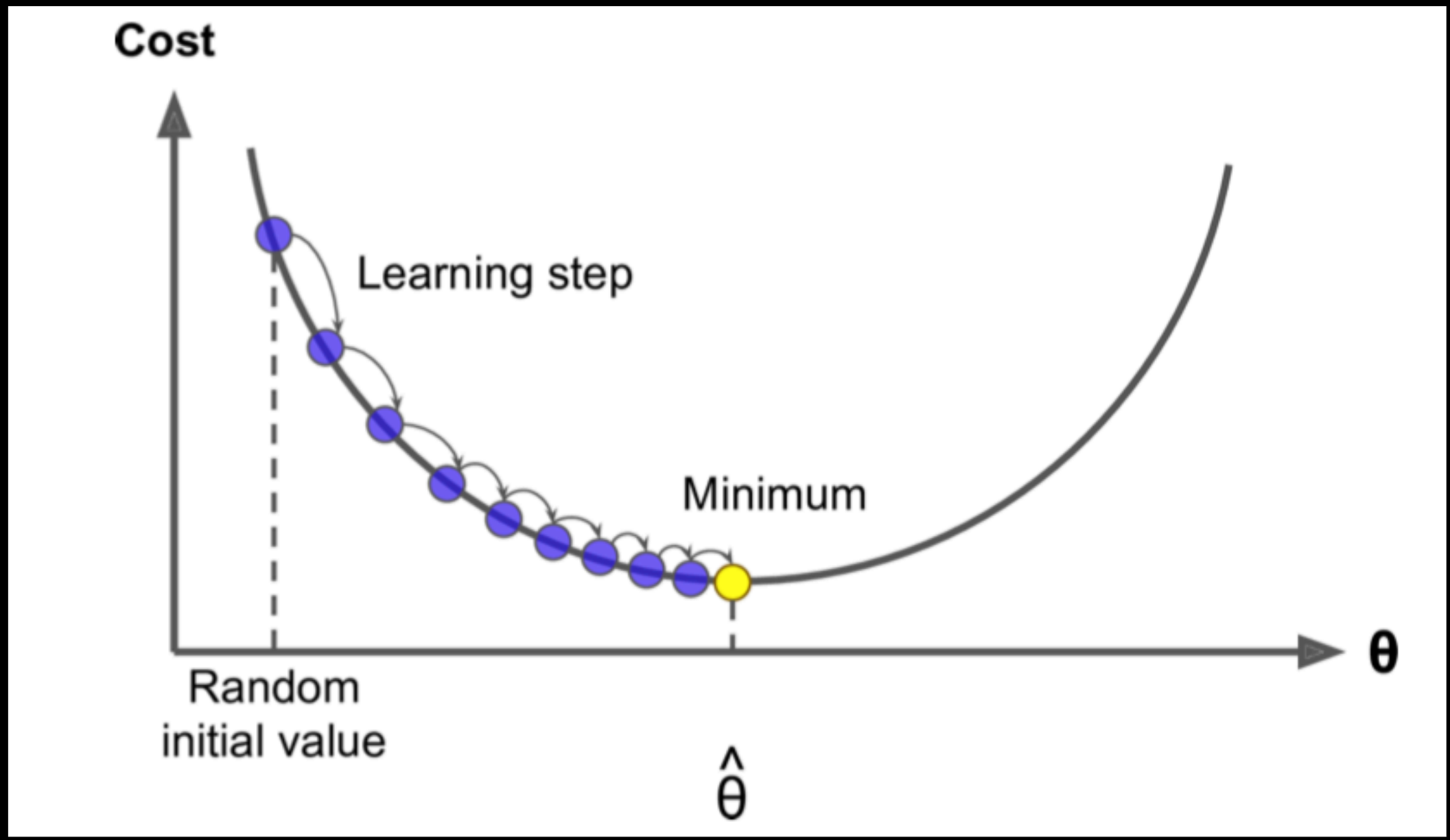
Gradient Descent

- GD mede o gradiente (taxa de mudança) local da função de custo/erro em relação ao vetor de coeficientes θ.

- E segue na direção do gradiente descendente.

- Uma vez que o gradiente (taxa de mudança) é zero, você atingiu um mínimo!

Gradient Descent



Gradient Descent

- Você começa preenchendo θ com valores aleatórios (isto é chamado de inicialização aleatória).

- Em seguida, você melhora gradualmente, tomando um pequeno passo de cada vez,

- Cada passo tentando diminuir a função de custo (por exemplo, o MSE) até que o algoritmo converge para um mínimo.

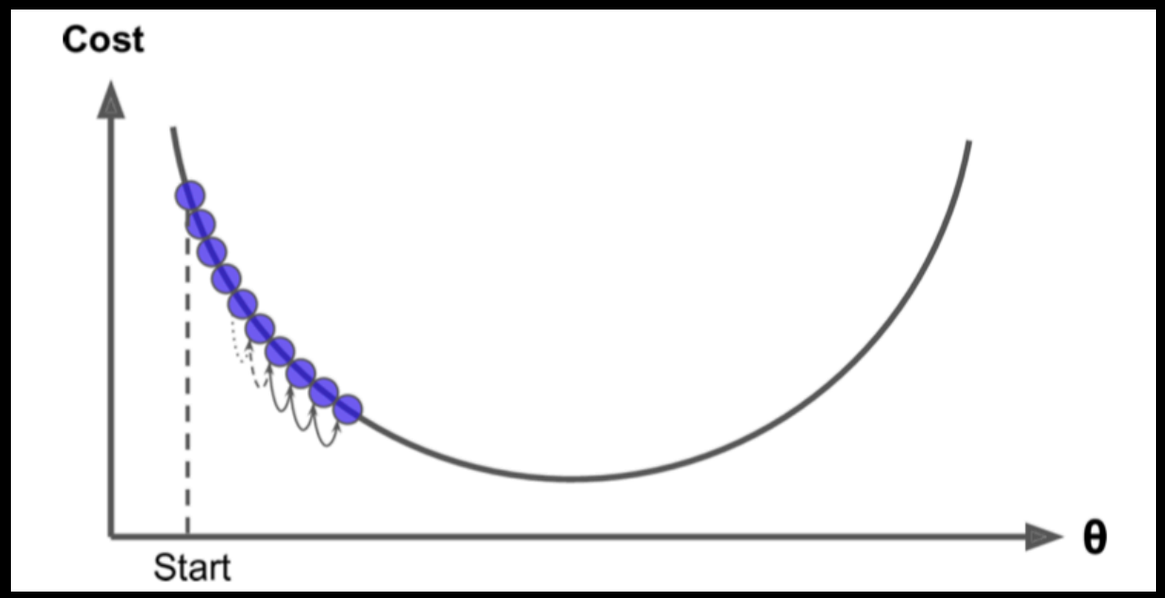
Gradient Descent (Descida de Gradiente)

- Um parâmetro importante em Gradient Descent é o tamanho das etapas, determinado pelo hiperparâmetro da taxa de aprendizado.

- Se a taxa de aprendizagem for muito pequena, então o algoritmo terá que passar por muitas iterações para convergir, o que levará muito tempo.

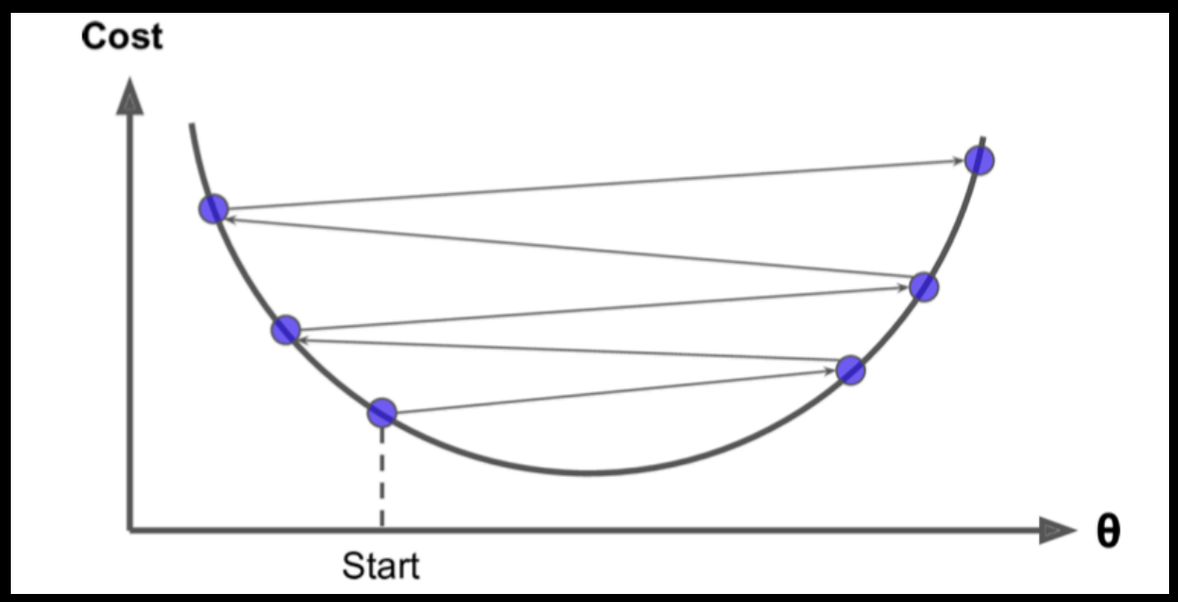
- Se ela for muito grande, o algoritmo vai dar grandes saltos de um lado para o outro e vai acabar divergindo, nunca encontrando uma boa solução.

Taxa de Aprendizagem

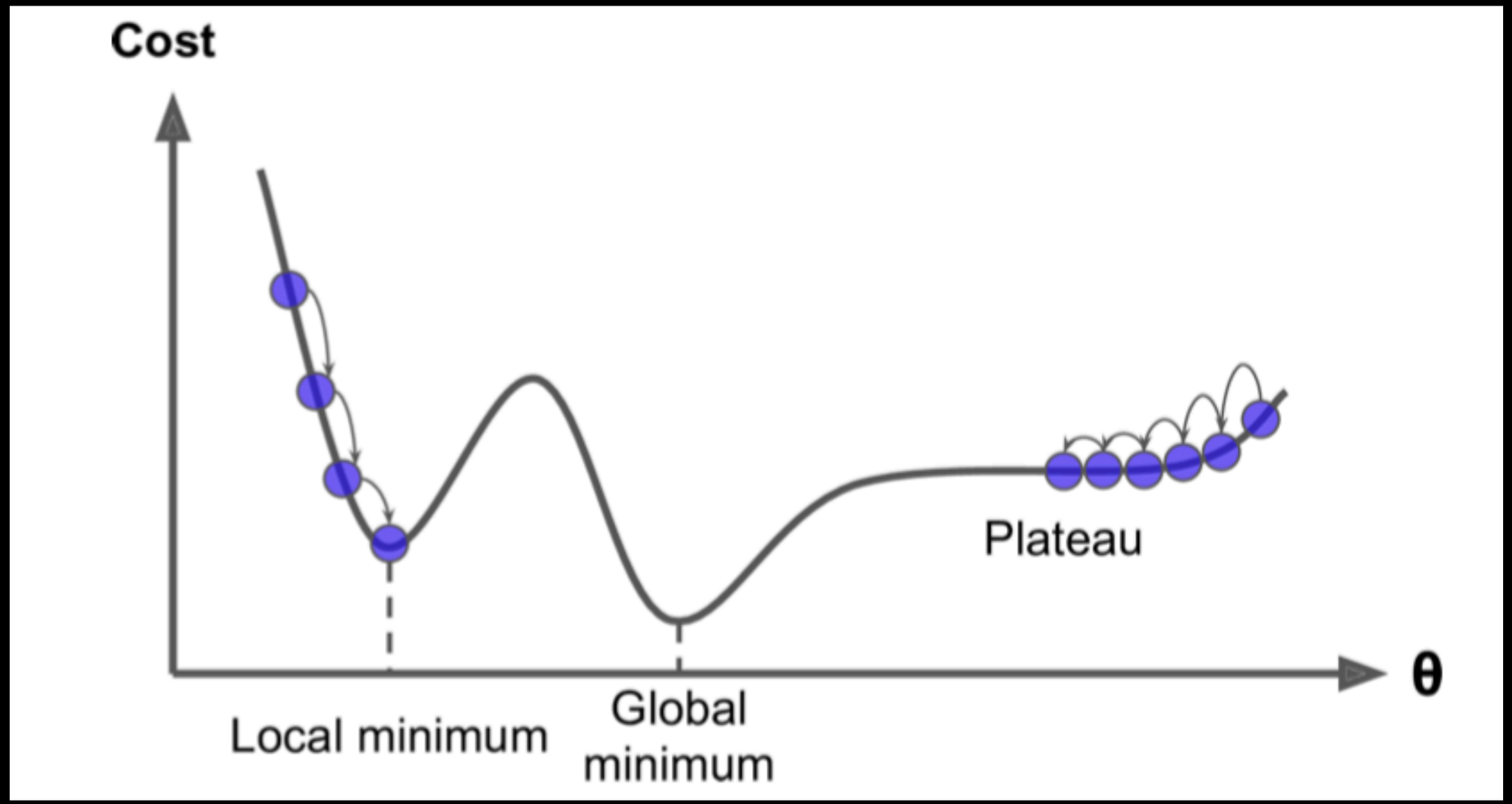


Pequena demais

Grande demais



Mínima Local

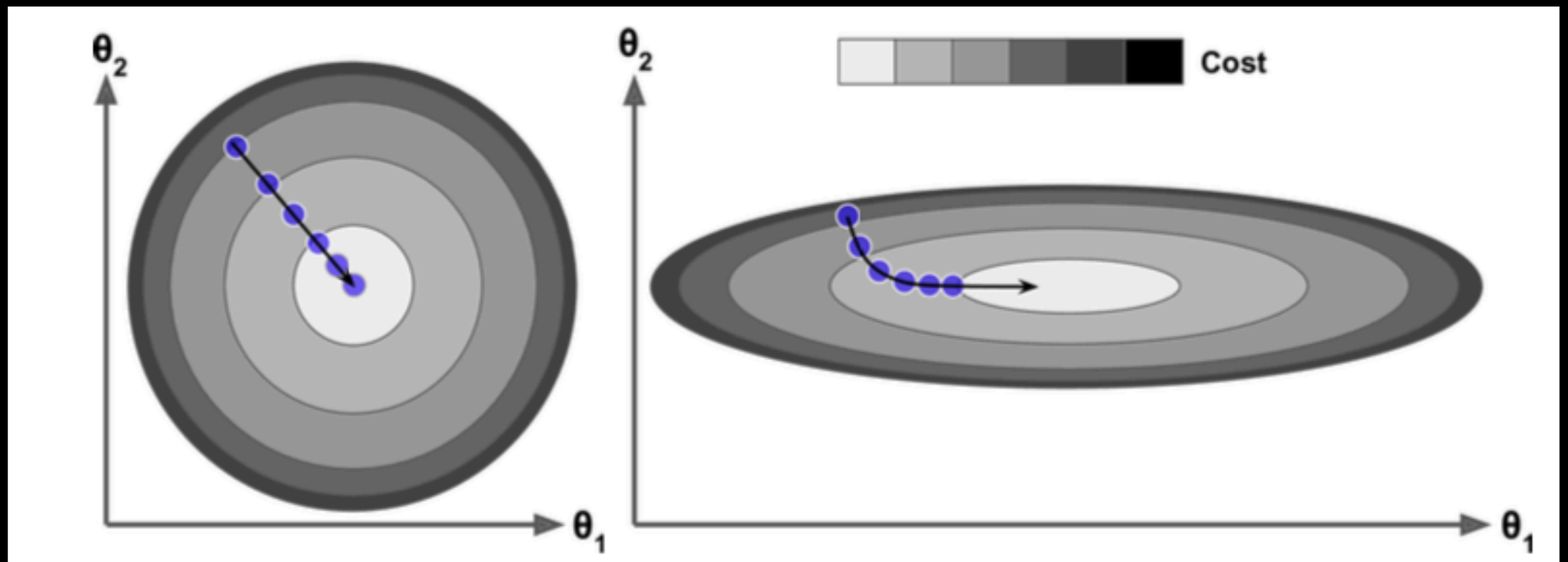


GD com Regressão Linear

- A função de custo de Regressão Linear é convexa.
- Quer dizer, se você selecionar dois pontos na curva, o segmento de linha que os une nunca cruza a curva. Isso implica que não há mínimos locais, apenas um mínimo global.
- É também uma função contínua com uma inclinação que nunca muda abruptamente.
- Esses dois fatos têm uma ótima conseqüência: Gradient Descent é garantido para se aproximar arbitrariamente do mínimo global (se você esperar o tempo suficiente e se a taxa de aprendizado não for muito alta) .

As Variáveis Independentes Devem Estar na Mesma Escala

Quando usar GD, se assegure que as variáveis independentes estão na mesma escala!



Variáveis independentes na mesma escala

Variáveis independentes não na mesma escala

Cálculo de GD

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

Derivada da função custo em relação a cada variável

$$\nabla_\theta \mathrm{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0}\mathrm{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1}\mathrm{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n}\mathrm{MSE}(\theta) \end{pmatrix} = \frac{2}{m}\mathbf{X}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y})$$

O vetor gradiente da função custo com todas as derivadas

n é a taxa de aprendizagem

Cada passo

$$\theta^{(\text{next step})} = \theta - \eta \nabla_\theta \mathrm{MSE}(\theta)$$

Calculando Descida de Gradiente em lote no Jupyter notebook

Parâmetros de GD

- Como configurar o número de iterações? Se for muito baixo, você ainda estará longe da solução ideal quando o algoritmo parar, mas se for muito alto, você perderá tempo enquanto os parâmetros do modelo não mudam mais.

- Uma solução simples é definir um número muito grande de iterações, mas interromper o algoritmo quando o vetor de gradiente se torna pequeno, ou seja, quando sua norma se torna menor do que um pequeno número ε (chamado de tolerância), porque isso ocorre quando o Gradiente Descent (quase) atingiu o mínimo.

- Para se encontrar uma boa taxa de aprendizagem, se pode usar Grid Search.

Tipos de GD

- GD de Lote (Batch) = é o que vimos até agora. Ele usa todo o conjunto de treinamento para calcular os gradientes em cada etapa. O que o torna muito lento quando o conjunto de treinamento é grande.

- GD Estocástico = No extremo oposto, Stochastic Gradient Descent escolhe apenas uma instância aleatória no conjunto de treinamento em cada etapa e calcula os gradientes com base apenas na única instância. Obviamente, isso torna o algoritmo muito mais rápido, pois tem dados muito pequenos para manipular em cada iteração. Também permite treinar em conjuntos de treinamento enormes, uma vez que apenas uma instância precisa estar na memória em cada iteração!

GD Estocástico

- Devido a sua natureza estocástica (ou seja, aleatória), esse algoritmo é muito menos regular do que GD de Lote: ao invés de diminuir suavemente até atingir o mínimo, a função de custo vai subir e descer, diminuindo apenas em média.

- Quando a função de custo é muito irregular, isso pode realmente ajudar o algoritmo a pular fora dos mínimos locais, de modo que a Descida de Gradiente Estocástico tem uma melhor chance de encontrar o mínimo global do que a Descida de Gradiente em Lote.

GD Estocástico

- Portanto, a aleatoriedade é boa para escapar da minima local, mas ruim porque significa que o algoritmo nunca pode se estabelecer no mínimo. Uma solução para este dilema é reduzir gradualmente a taxa de aprendizado.

- As etapas começam grandes (o que ajuda a fazer progressos rápidos e a escapar dos mínimos locais) e, em seguida, cada vez menor, permitindo que o algoritmo se assente no mínimo global.

- Este processo é chamado de simulated annealing, porque se assemelha ao processo de resfriamento na metalurgia onde o metal fundido é esfriado lentamente. A função que determina a taxa de aprendizagem em cada iteração é chamada de cronograma de aprendizado.

Calculando Descida de Gradiente Estocástico no Jupyter notebook

Sumário:
Algoritmos de Regressão Linear

	Muitas instâncias, observações	Muitas variáveis independentes	Variáveis na mesma escala	Classe na Scikit-Learn
Descida de Gradiente Estocástica (SGD)	Rápido	Rápido	Sim	SGDRegressor
Descida de Gradiente de Lote (BGD)	Lento	Rápido	Sim	Nenhuma
Solução pela equação de minimização	Rápido	Lento	Não	LinearRegression

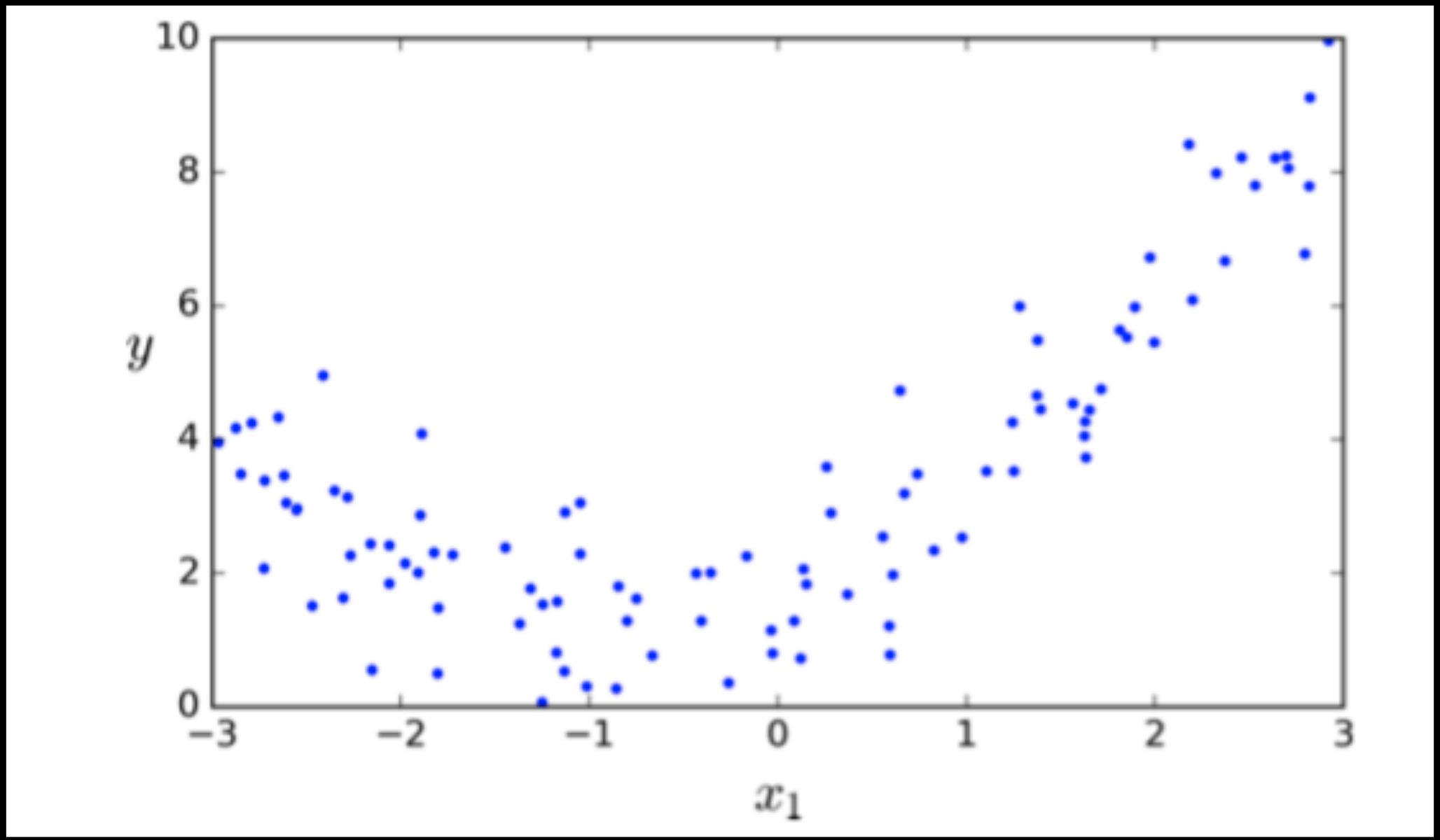
Exercício Prático 1:

Regressão Linear

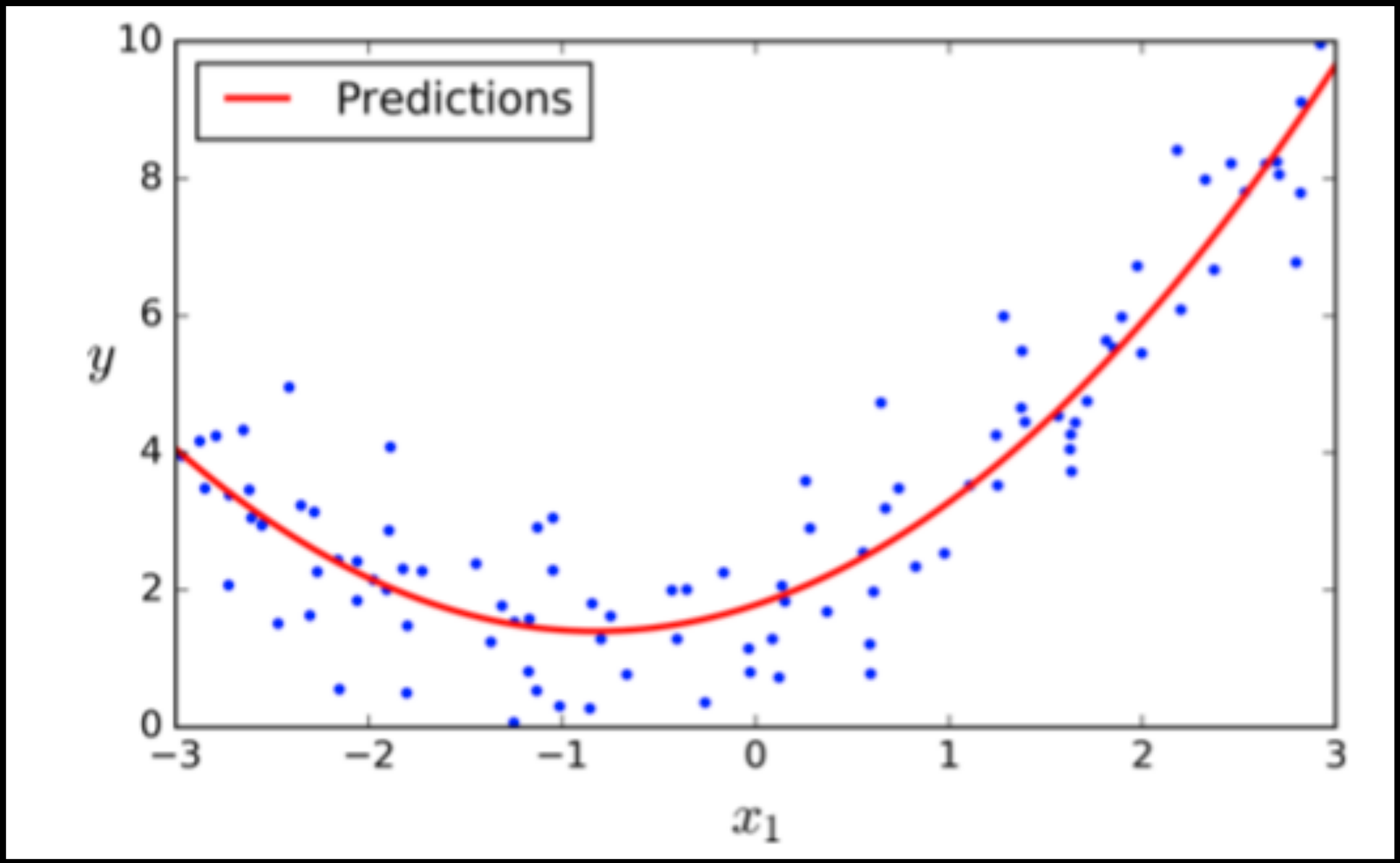
Regressão Polinomial

- Podemos usar também Regressão Linear quando o relacionamento entre as variáveis independentes e a variável objetivo (a dependente) não é “linear”,
- Este modelo funciona também bem em relacionamentos não lineares,
- Se o relacionamento for proporcional!

Regressão Polinomial



Regressão Polinomial



Regressão Polinomial

- Podemos adicionar graus mais altos a cada variáveis como segundo ou terceiro grau.
- Em Python temos para isto a classe PolynomialFeatures.
- Esta classe faz pré-processamento!
- Transformando os dados, adicionando, neste caso, 2o. grau seria bom, o quadrado de cada variável no conjunto de treinamento.

Exemplo
no Jupyter notebook

Exercício Prático 2:

Regressão Linear