Aula 03

Gradiente Descendente e Otimização de Funções

Agenda

01

Recapitulação: Função de Custo

02

Minimização da Função de Custo

03

Gradiente Descendente

04

Equação Normal

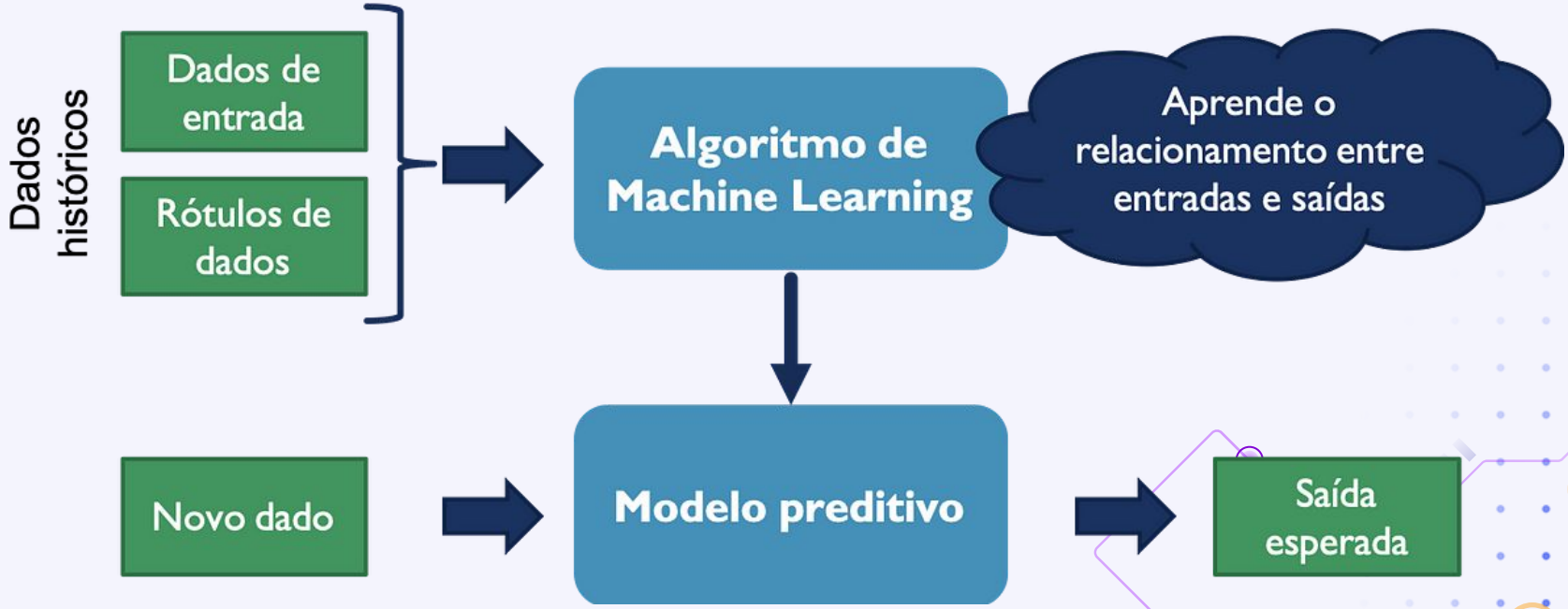
05

Implementando os algoritmos

01

Recapitulação: Função de Custo

Modelando um fenômeno



Retirado de: https://tatianaesc.medium.com/machine-learning-conceitos-e-modelos-f0373bf4f445

Quão bom é o nosso modelo?

Como esse ajuste é calculado?

Podemos metrificar o quão bom o nosso modelo é?

Qual é o **erro** dele?

O erro do modelo é uma espécie de diferença entre o chute do modelo e a resposta correta.

Quão bom é o nosso modelo?

Como esse ajuste é calculado?

Vamos pensar no nosso erro como **um custo** ou **uma perda**.

Queremos minimizar o custo!

Objetivo: minimizar uma função que representa o custo do nosso modelo.

O que é uma Regressão Linear?

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$$

Uma função F é linear se:
*F(x + y) = F(x) + F(y) e c*F(x) = F(c*x)*

Pode ser representada por uma reta, um plano, . . .

Qual a Função de Custo?

A nossa métrica de erro será a *Mean Squared Error*:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

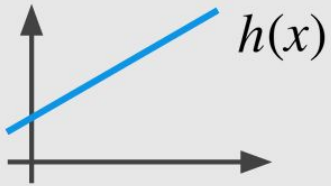
Função de Custo

Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Parameters:

$$\theta_0, \theta_1$$



Cost Function:

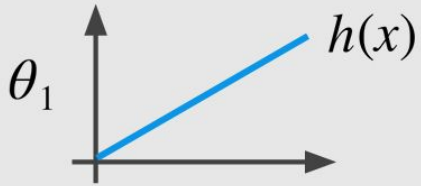
$$J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Goal:

$$\underset{\theta_0,\theta_1}{\text{minimize}}\ J(\theta_0,\theta_1)$$

Simplified

$$h_\theta(x) = \theta_1 x$$



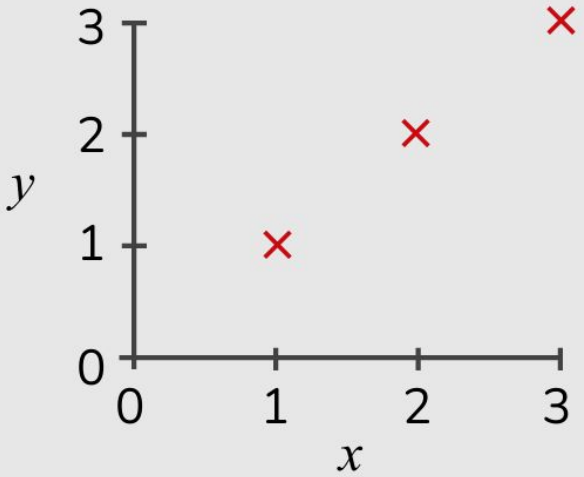
$$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\underset{\theta_1}{\text{minimize}}\ J(\theta_1)$$

Função de Custo

$h_\theta(x)$

(for fixed θ_1, this is a function of x)



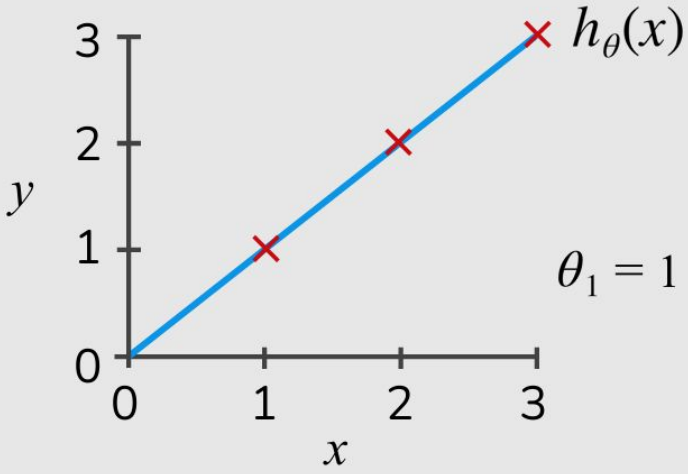
$J(\theta_1)$

(function of the parameters θ_1)

Função de Custo

$h_\theta(x)$

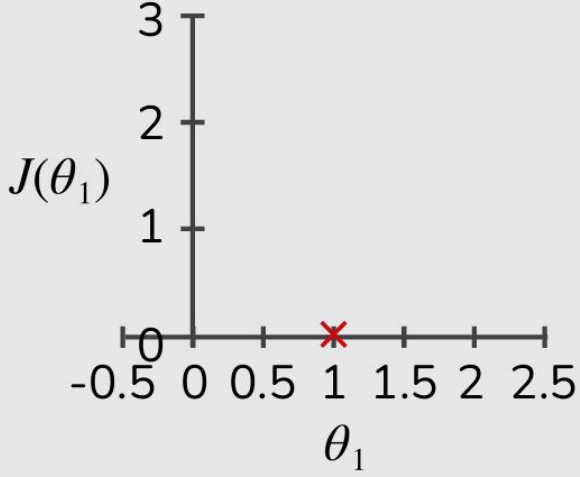
(for fixed θ_1, this is a function of x)



$J(\theta_1) = J(1) = 0$

$J(\theta_1)$

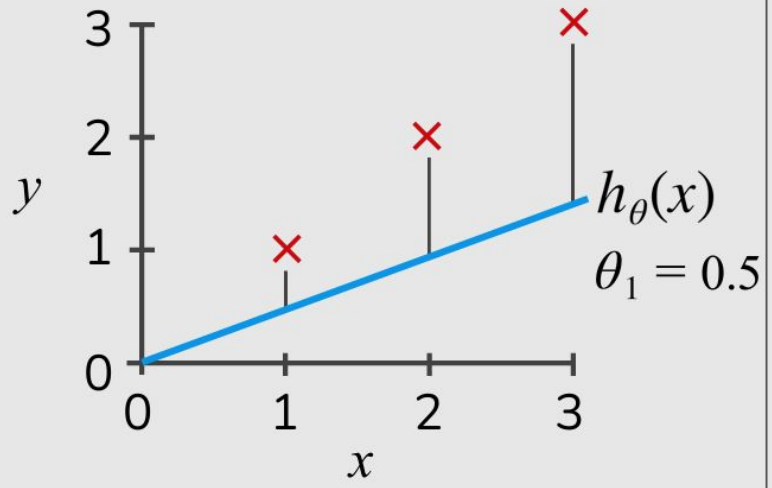
(function of the parameters θ_1)



Função de Custo

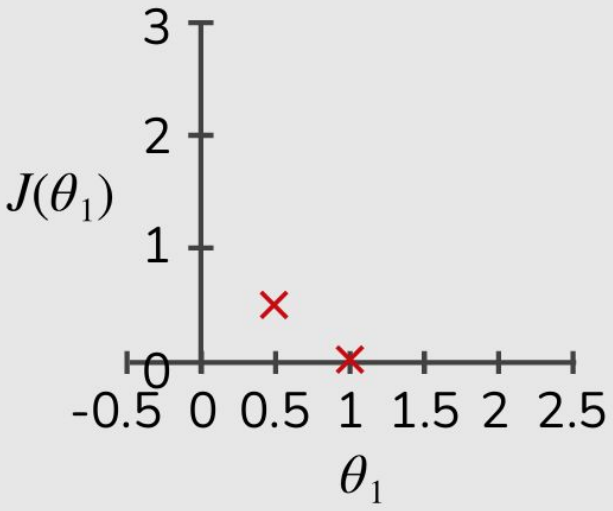
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

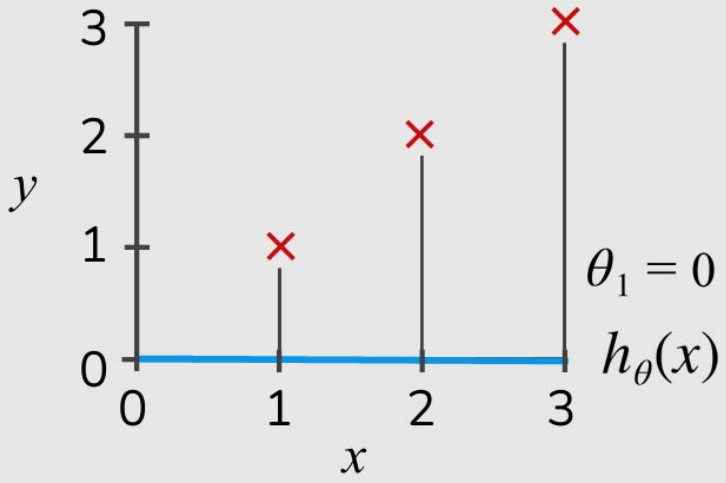
(function of the parameters θ_1)



Função de Custo

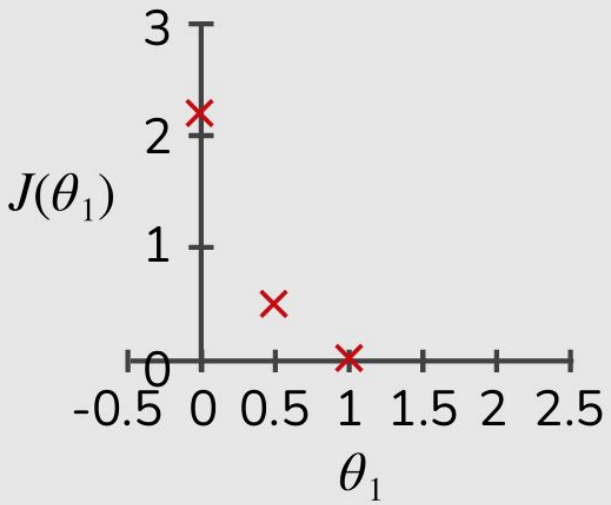
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

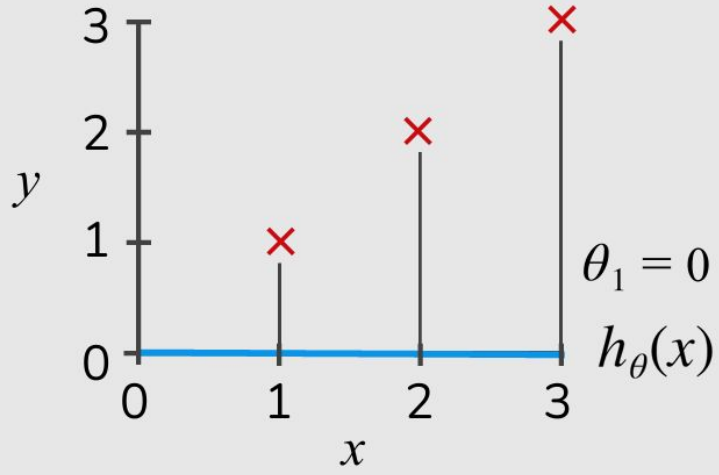
(function of the parameters θ_1)



Função de Custo

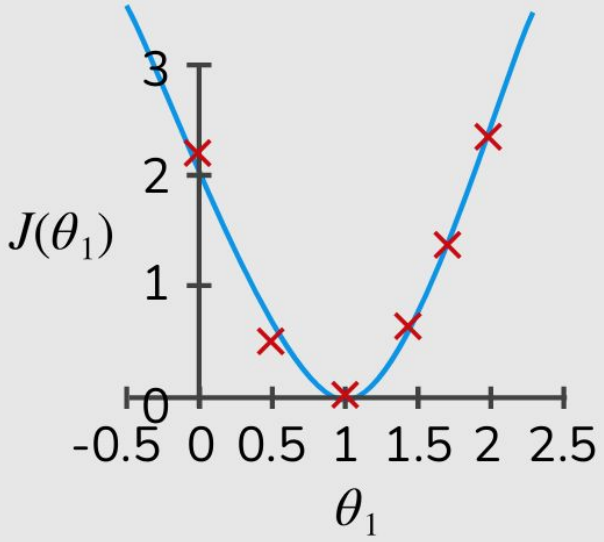
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

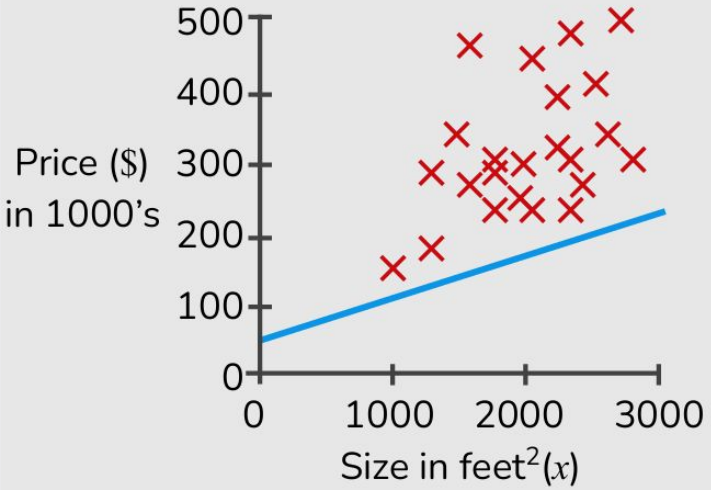
(function of the parameters θ_1)



Função de Custo

$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



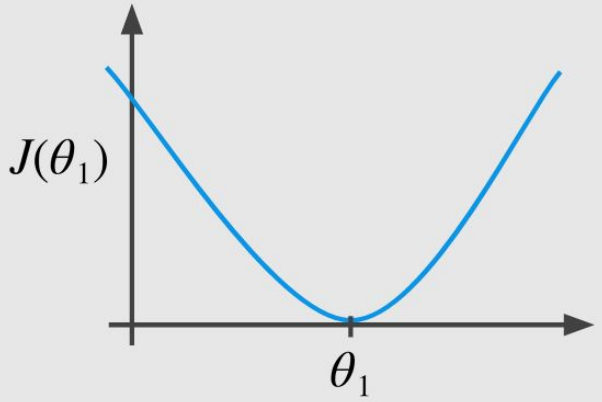
$h_\theta(x) = 50 + 0.06x$

$\theta_0 = 50$

$\theta_1 = 0.06$

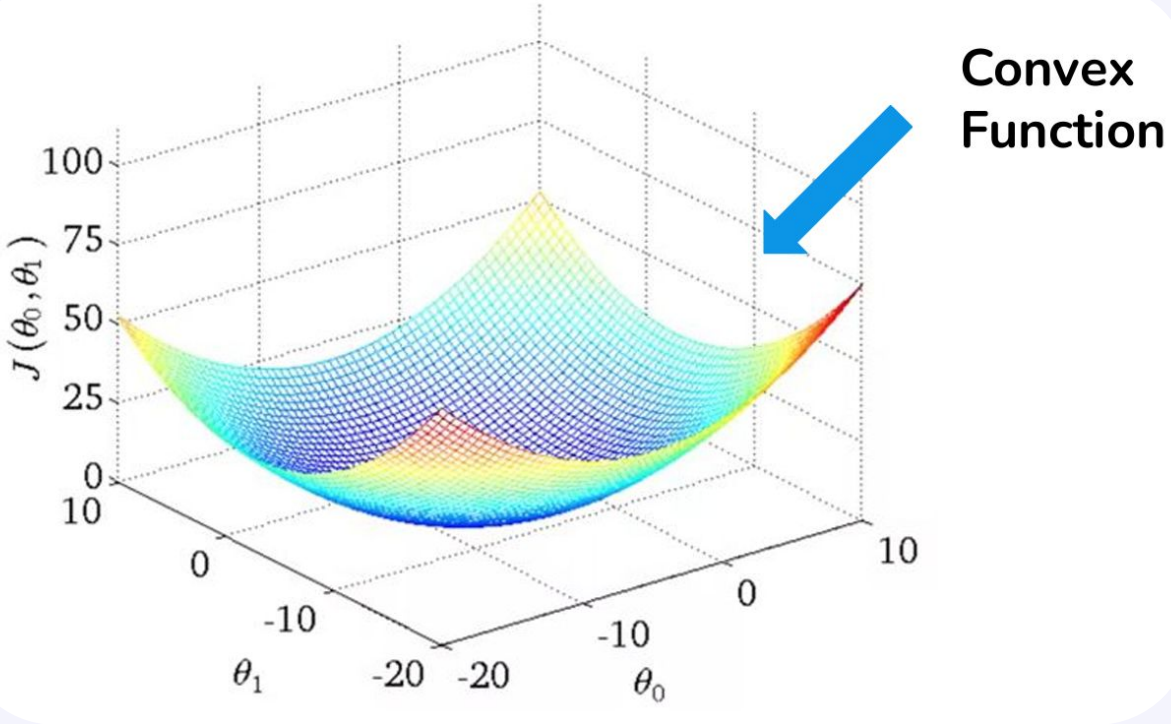
$J(\theta_0,\theta_1)$

(function of the parameters θ_0, θ_1)



θ_0 and θ_1?

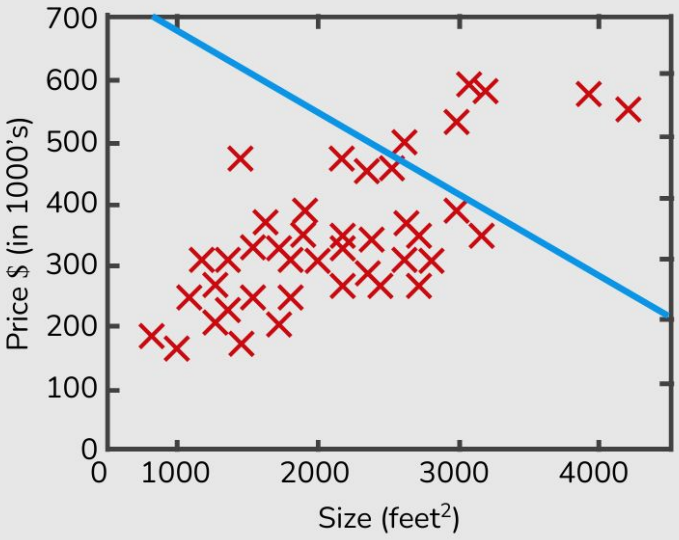
Função de Custo



Função de Custo

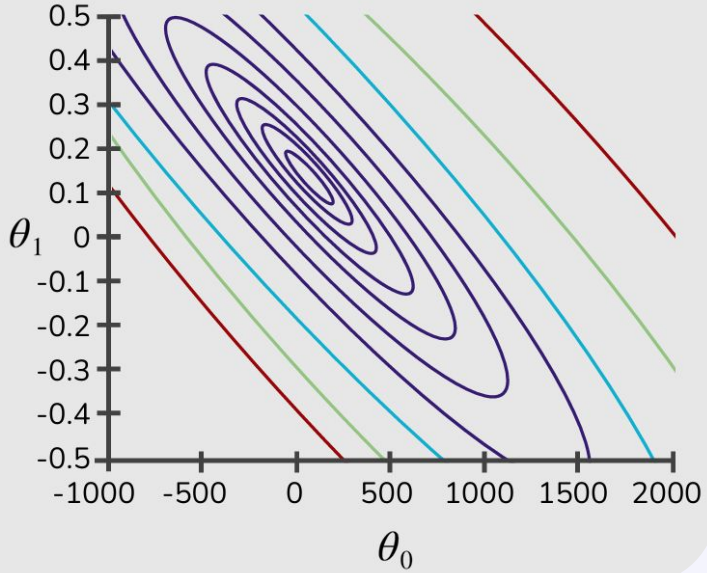
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

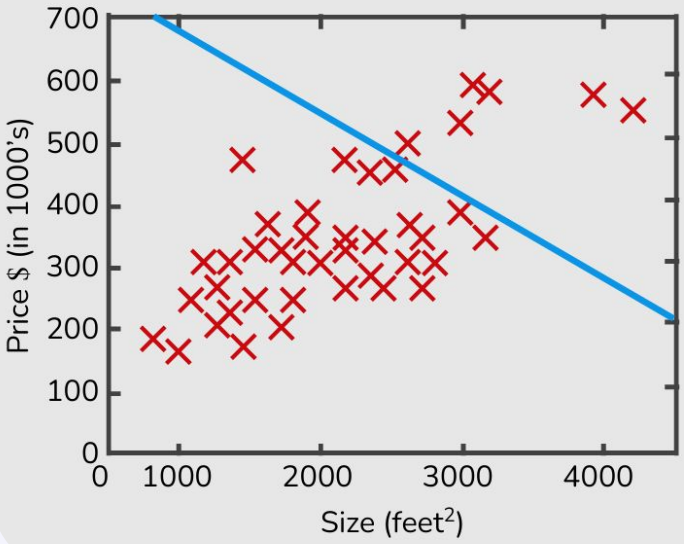
(function of the parameters θ_0, θ_1)



Função de Custo

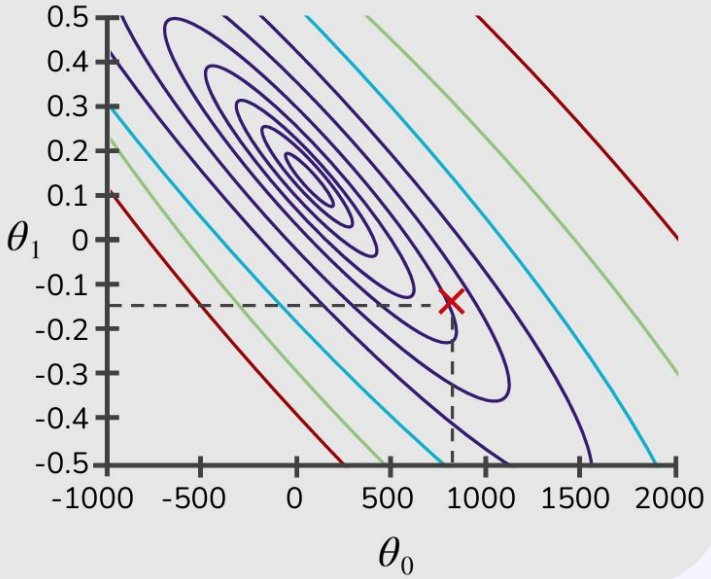
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

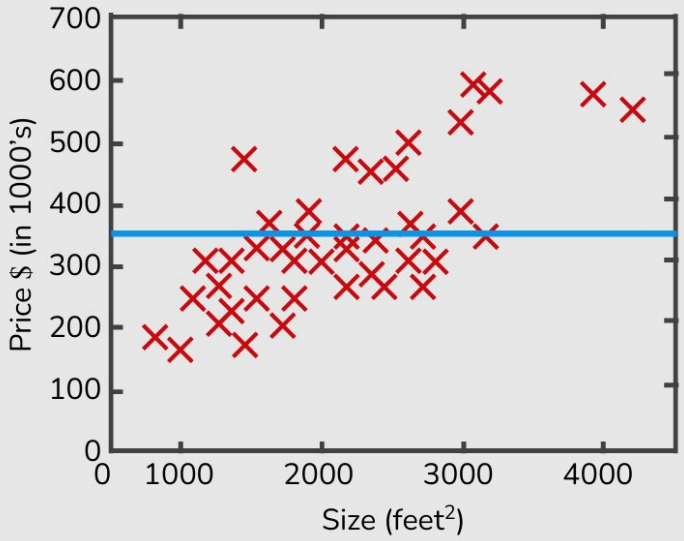
(function of the parameters θ_0, θ_1)



Função de Custo

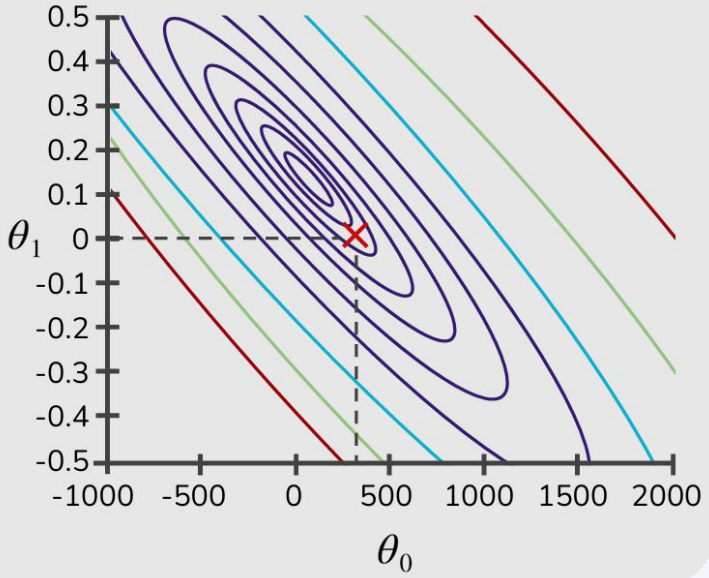
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

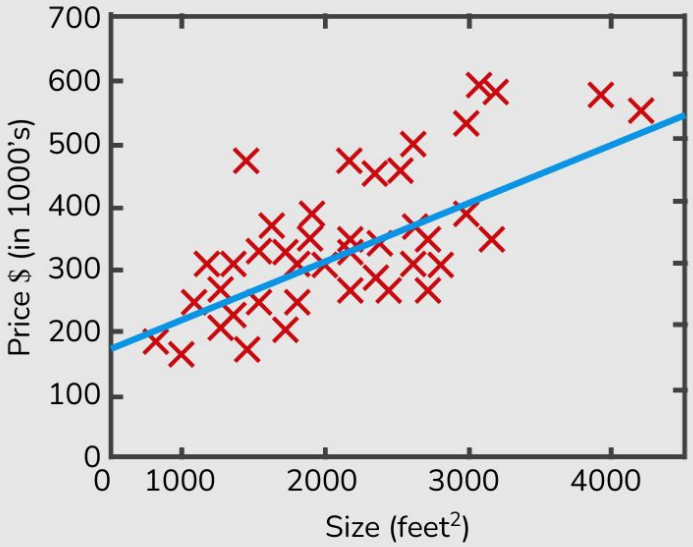
(function of the parameters θ_0, θ_1)



Função de Custo

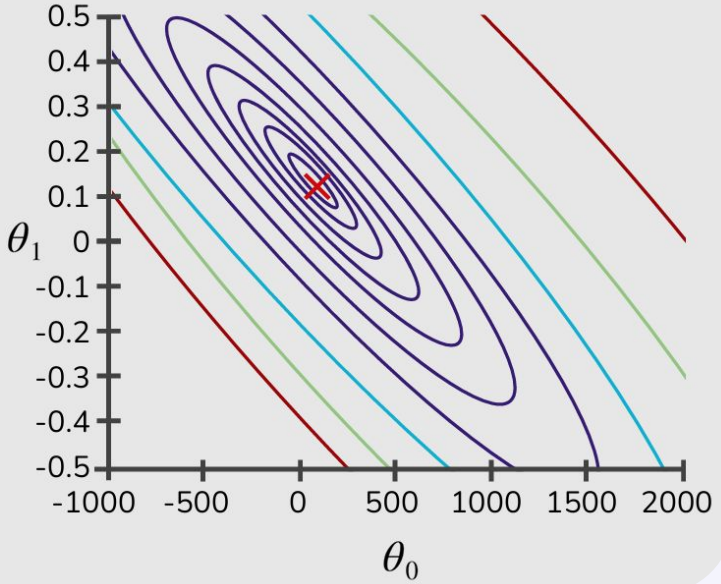
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



02

Minimização da Função de Custo

O que é minimizar uma função?

Minimizar uma função é encontrar os valores de entrada que retornam **o menor valor possível** para aquela função.

Este valor também é conhecido como o **mínimo global**.

Achar o mínimo global **nem sempre é possível**.

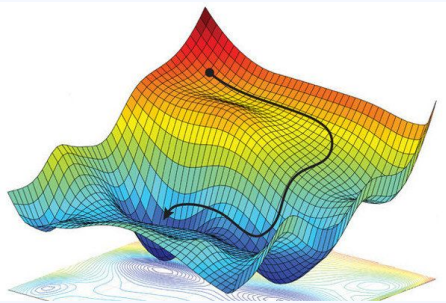
Às vezes, é possível encontrar apenas **mínimos locais**.

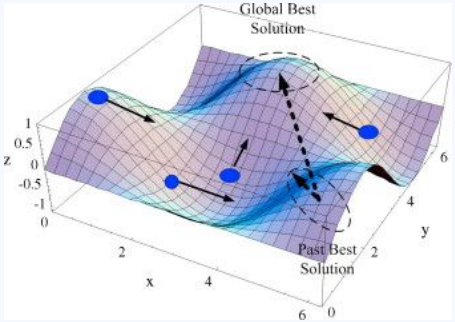
Encontrar **o mínimo** ou **o máximo** de funções são **problemas de otimização**.

Alguns algoritmos de otimização

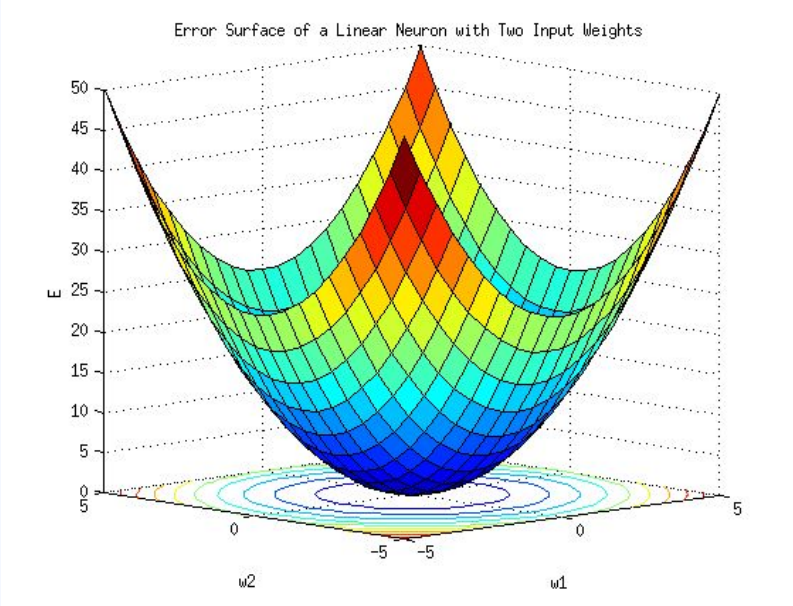
- MMQ → Equação Normal
  (*OLS*) (*Normal Equation*)
- **Gradiente Descendente**
  (*Gradient Descent*)
- Otimização por Enxame de Partículas
  (*Particle Swarm Optimization*)
- Algoritmo Genético
  (*Genetic Algorithm*)

$$\hat{\beta} = \left(X^1 X\right)^{-1} X^1 Y$$



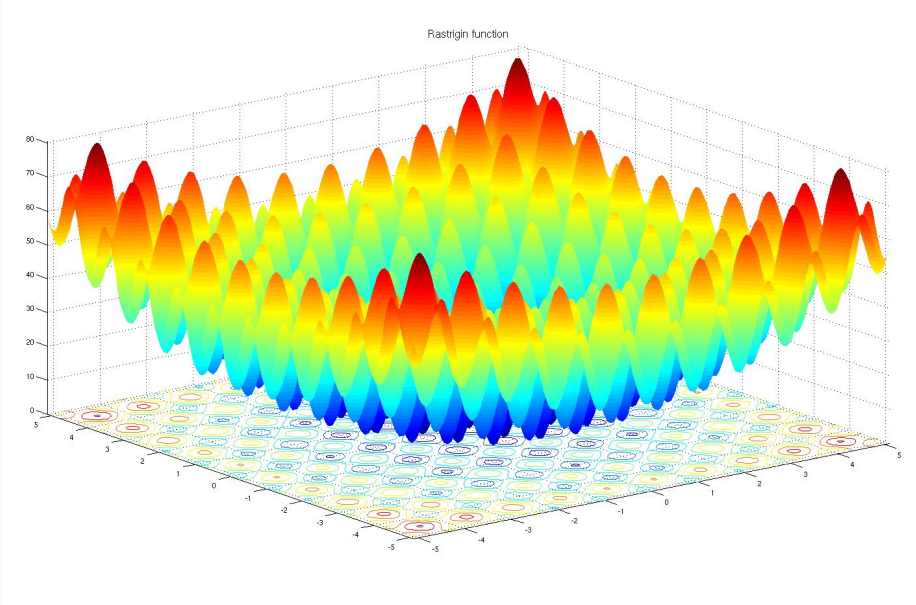


Alguns algoritmos de otimização



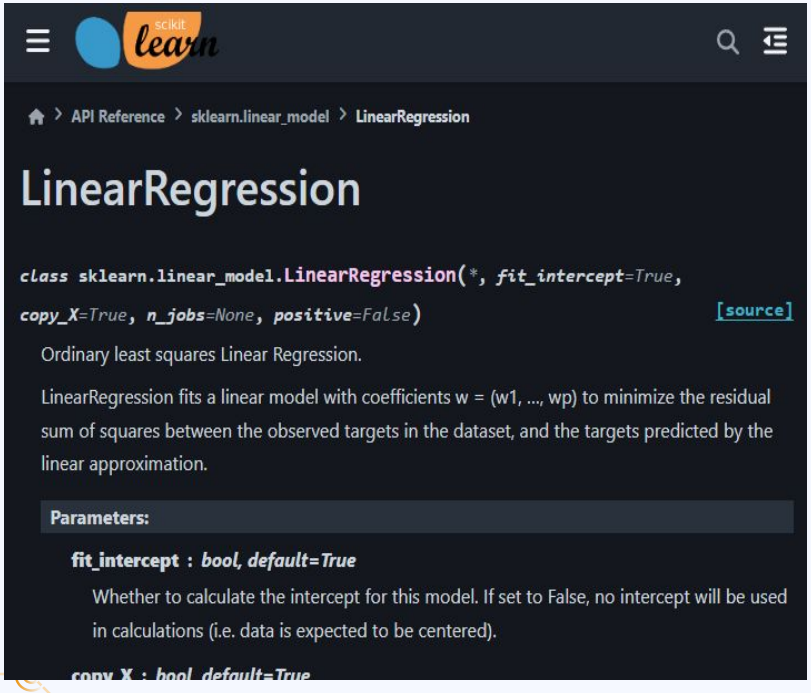
Função de Custo em forma de parabolóide
→ **MSE** usado em Regressão Linear

Função de benchmark (Rastrigin) de proc. otimização



Alguns algoritmos de otimização

Exemplo: a classe `LinearRegression` do *sklearn*.



scikit learn

API Reference › sklearn.linear_model › LinearRegression

LinearRegression

class **sklearn.linear_model.LinearRegression**(*, *fit_intercept*=*True*, *copy_X*=*True*, *n_jobs*=*None*, *positive*=*False*) [source]

Ordinary least squares Linear Regression.

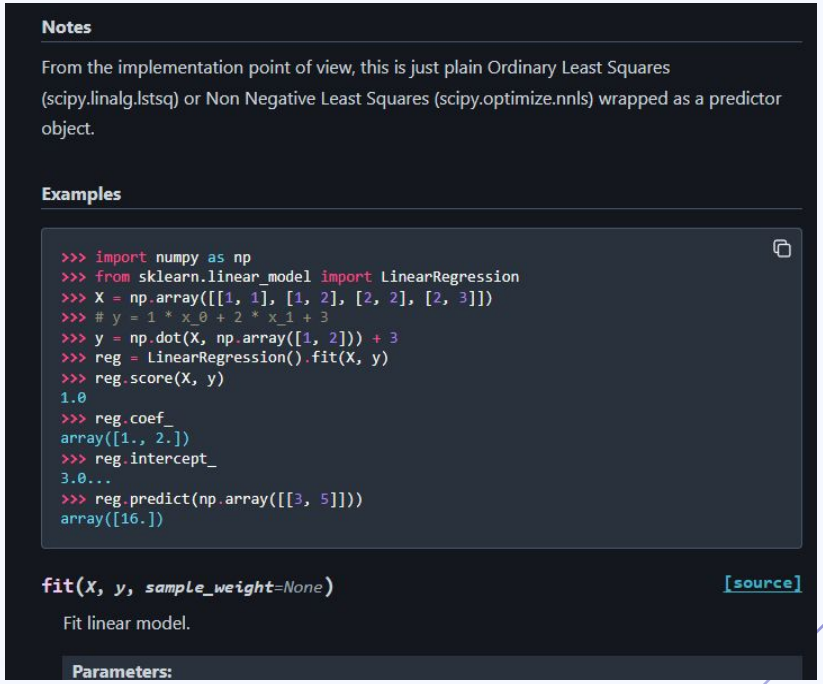
LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Parameters:

fit_intercept : *bool, default=True*

Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

copy_X : *bool, default=True*



Notes

From the implementation point of view, this is just plain Ordinary Least Squares (scipy.linalg.lstsq) or Non Negative Least Squares (scipy.optimize.nnls) wrapped as a predictor object.

Examples

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> # y = 1 * x_0 + 2 * x_1 + 3
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)
>>> reg.score(X, y)
1.0
>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

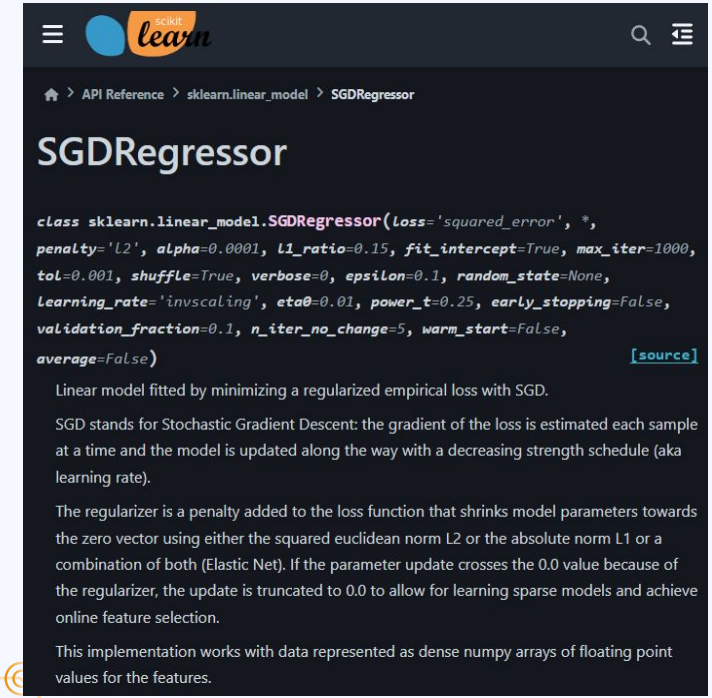
fit(*X*, *y*, *sample_weight*=*None*) [source]

Fit linear model.

Parameters:

Alguns algoritmos de otimização

Exemplo: a classe `SGDRegressor` do *sklearn*.



scikit-learn

API Reference > sklearn.linear_model > SGDRegressor

SGDRegressor

```
class sklearn.linear_model.SGDRegressor(loss='squared_error', *,
penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000,
tol=0.001, shuffle=True, verbose=0, epsilon=0.1, random_state=None,
learning_rate='invscaling', eta0=0.01, power_t=0.25, early_stopping=False,
validation_fraction=0.1, n_iter_no_change=5, warm_start=False,
average=False)
```

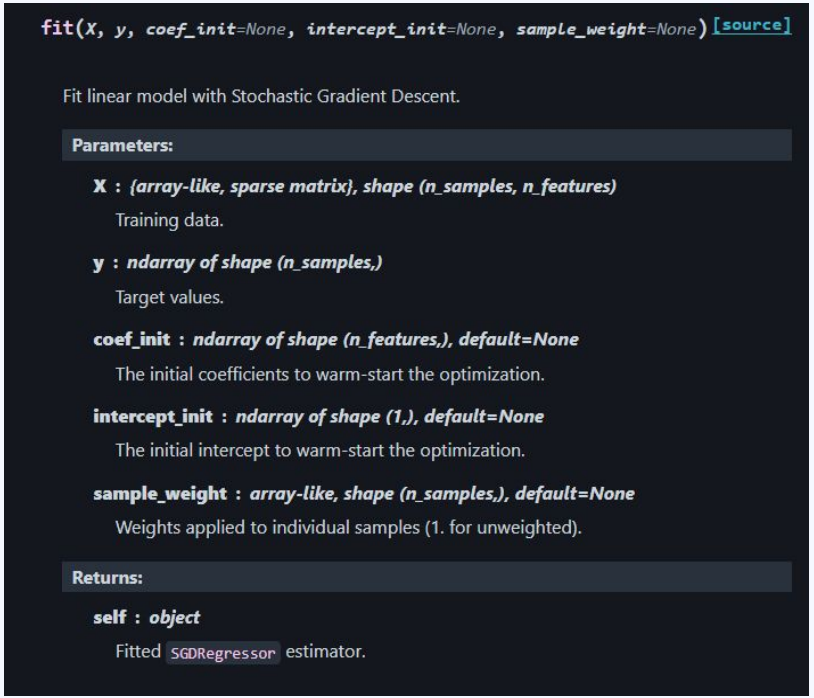
[source]

Linear model fitted by minimizing a regularized empirical loss with SGD.

SGD stands for Stochastic Gradient Descent: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate).

The regularizer is a penalty added to the loss function that shrinks model parameters towards the zero vector using either the squared euclidean norm L2 or the absolute norm L1 or a combination of both (Elastic Net). If the parameter update crosses the 0.0 value because of the regularizer, the update is truncated to 0.0 to allow for learning sparse models and achieve online feature selection.

This implementation works with data represented as dense numpy arrays of floating point values for the features.



```
fit(X, y, coef_init=None, intercept_init=None, sample_weight=None) [source]
```

Fit linear model with Stochastic Gradient Descent.

Parameters:

X : *{array-like, sparse matrix}, shape (n_samples, n_features)*

Training data.

y : *ndarray of shape (n_samples,)*

Target values.

coef_init : *ndarray of shape (n_features,), default=None*

The initial coefficients to warm-start the optimization.

intercept_init : *ndarray of shape (1,), default=None*

The initial intercept to warm-start the optimization.

sample_weight : *array-like, shape (n_samples,), default=None*

Weights applied to individual samples (1. for unweighted).

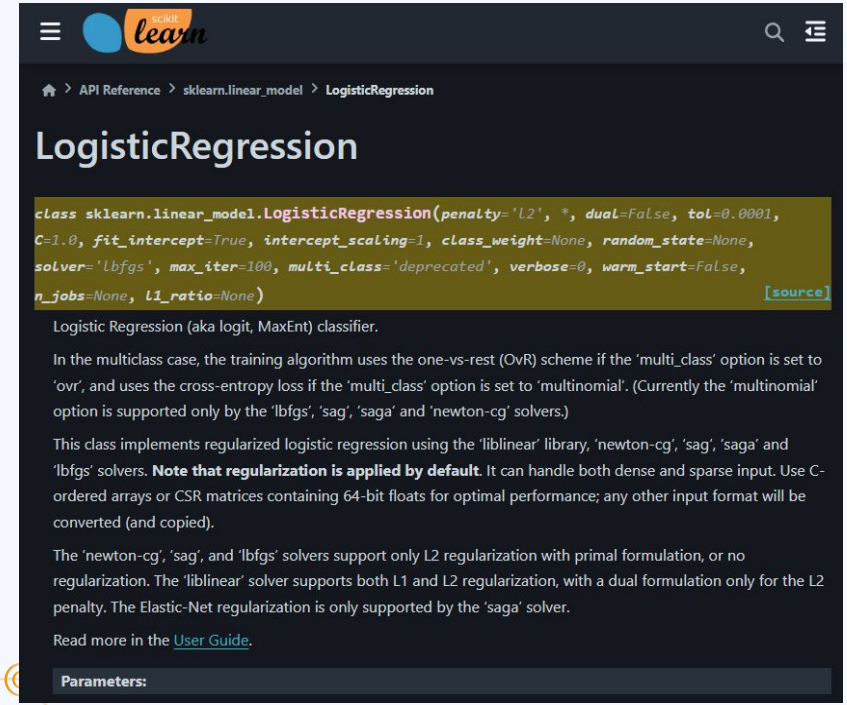
Returns:

self : *object*

Fitted `SGDRegressor` estimator.

Alguns algoritmos de otimização

Exemplo: a classe `LogisticRegression` do *sklearn*.



scikit-learn

API Reference > sklearn.linear_model > LogisticRegression

LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001,
C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None,
solver='lbfgs', max_iter=100, multi_class='deprecated', verbose=0, warm_start=False,
n_jobs=None, l1_ratio=None)                                                    [source]
```

Logistic Regression (aka logit, MaxEnt) classifier.

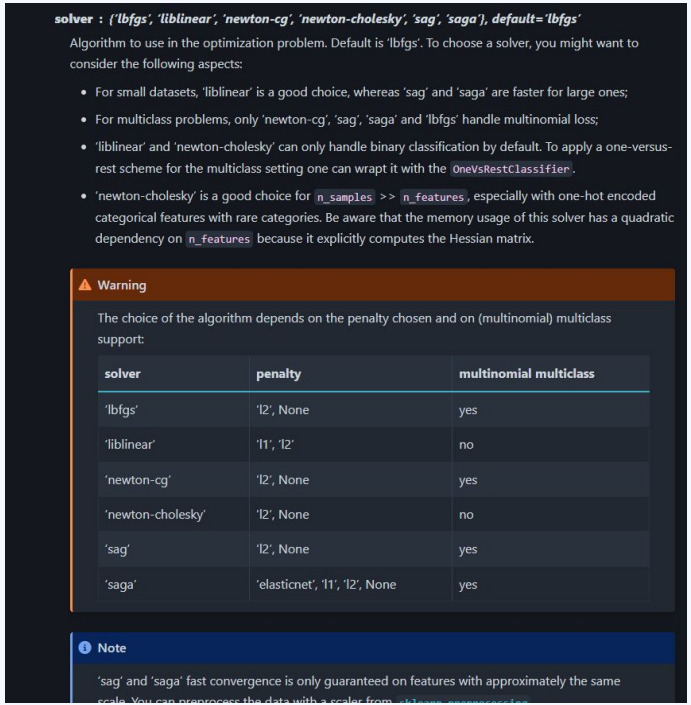
In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. **Note that regularization is applied by default**. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation, or no regularization. The 'liblinear' solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty. The Elastic-Net regularization is only supported by the 'saga' solver.

Read more in the User Guide.

Parameters:



solver : {'lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'}, default='lbfgs'

Algorithm to use in the optimization problem. Default is 'lbfgs'. To choose a solver, you might want to consider the following aspects:

- For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones;
- For multiclass problems, only 'newton-cg', 'sag', 'saga' and 'lbfgs' handle multinomial loss;
- 'liblinear' and 'newton-cholesky' can only handle binary classification by default. To apply a one-versus-rest scheme for the multiclass setting one can wrap it with the `OneVsRestClassifier`.
- 'newton-cholesky' is a good choice for `n_samples` >> `n_features`, especially with one-hot encoded categorical features with rare categories. Be aware that the memory usage of this solver has a quadratic dependency on `n_features` because it explicitly computes the Hessian matrix.

Warning

The choice of the algorithm depends on the penalty chosen and on (multinomial) multiclass support:

solver	penalty	multinomial multiclass
'lbfgs'	'l2', None	yes
'liblinear'	'l1', 'l2'	no
'newton-cg'	'l2', None	yes
'newton-cholesky'	'l2', None	no
'sag'	'l2', None	yes
'saga'	'elasticnet', 'l1', 'l2', None	yes

Note

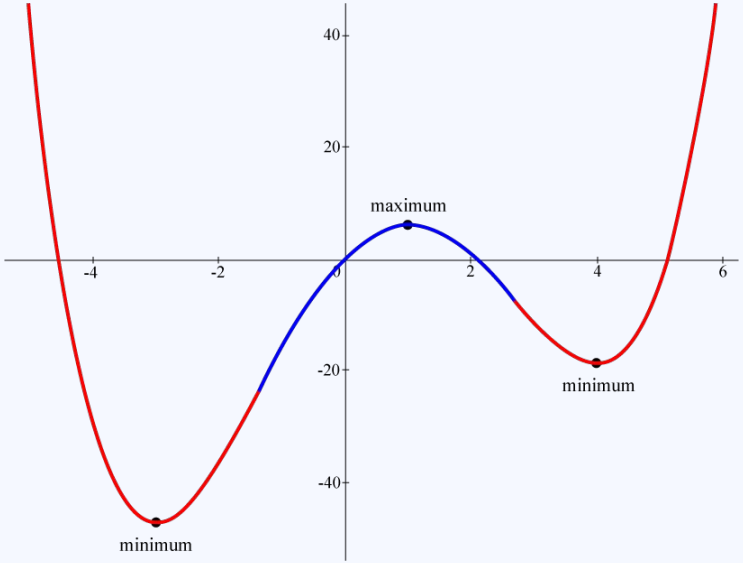
'sag' and 'saga' fast convergence is only guaranteed on features with approximately the same scale. You can preprocess the data with a scaler from sklearn.preprocessing.

03

Gradiente Descendente

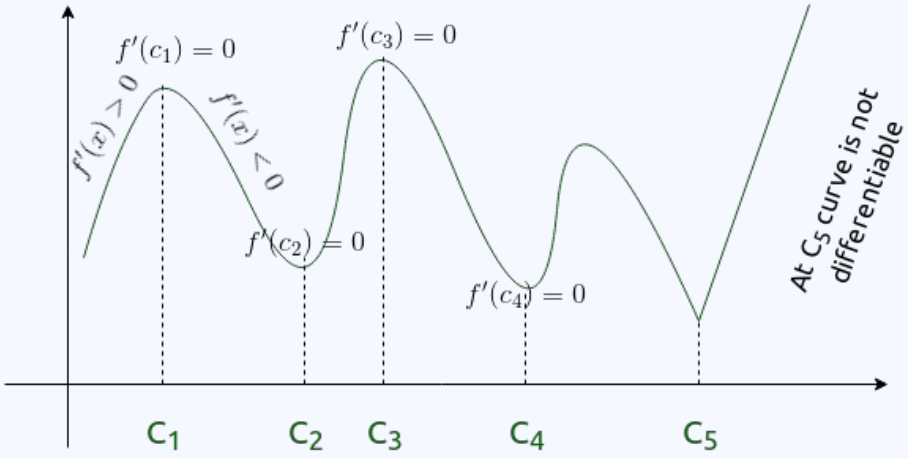
Relembrando derivadas

Como minimizamos funções em Cálculo Diferencial?



Calcworkshop.com

Relembrando derivadas



Derivada:

"Taxa" de variação de uma função em certo ponto.

Mínimo local:

Derivada de primeira ordem é igual a 0.

Testes de derivada de primeira e segunda ordem.

Relembrando derivadas

Vetor Gradiente:

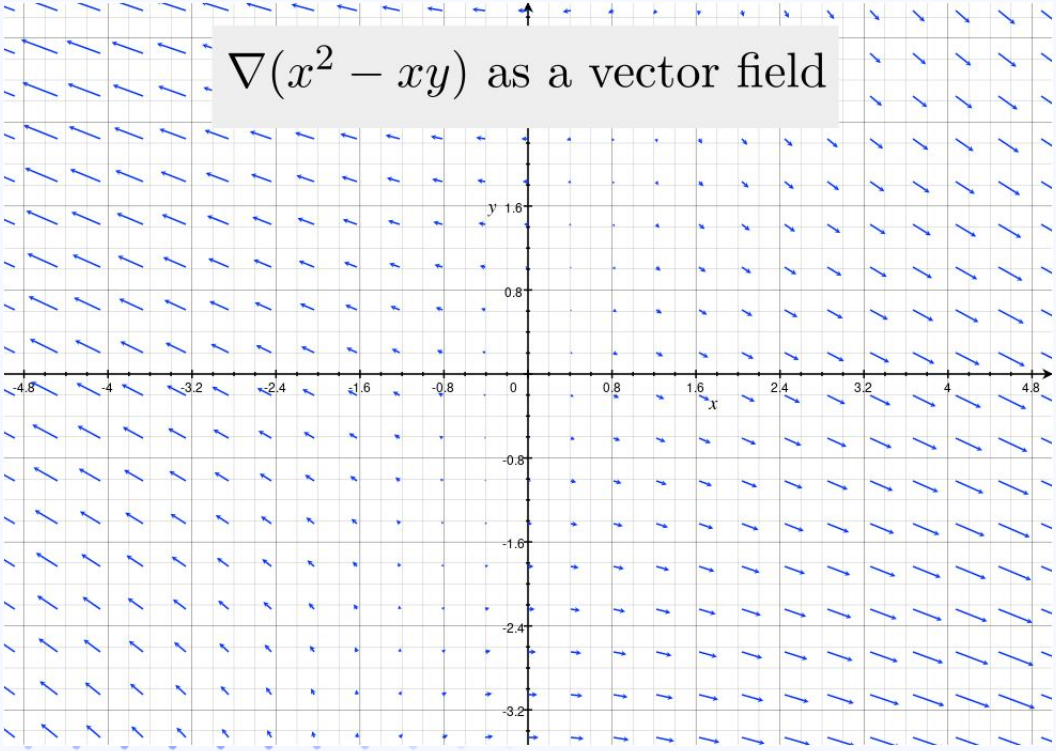
$$\nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z})$$

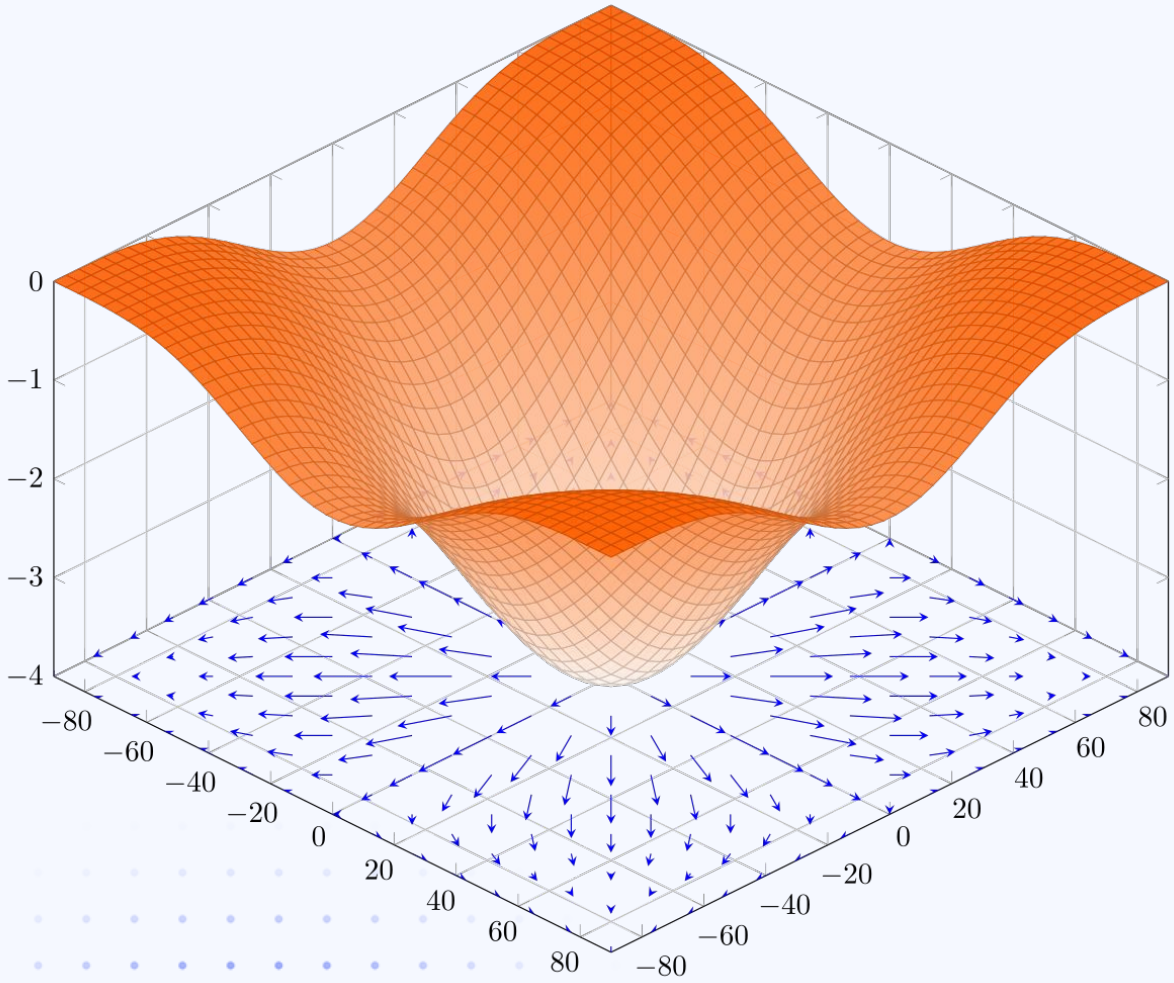
$$F(x, y, z) = x + y^2 + z^3$$

$$\nabla F(x, y, z) = (\frac{dF}{dx}, \frac{dF}{dy}, \frac{dF}{dz}) = (1, 2y, 3z^2)$$

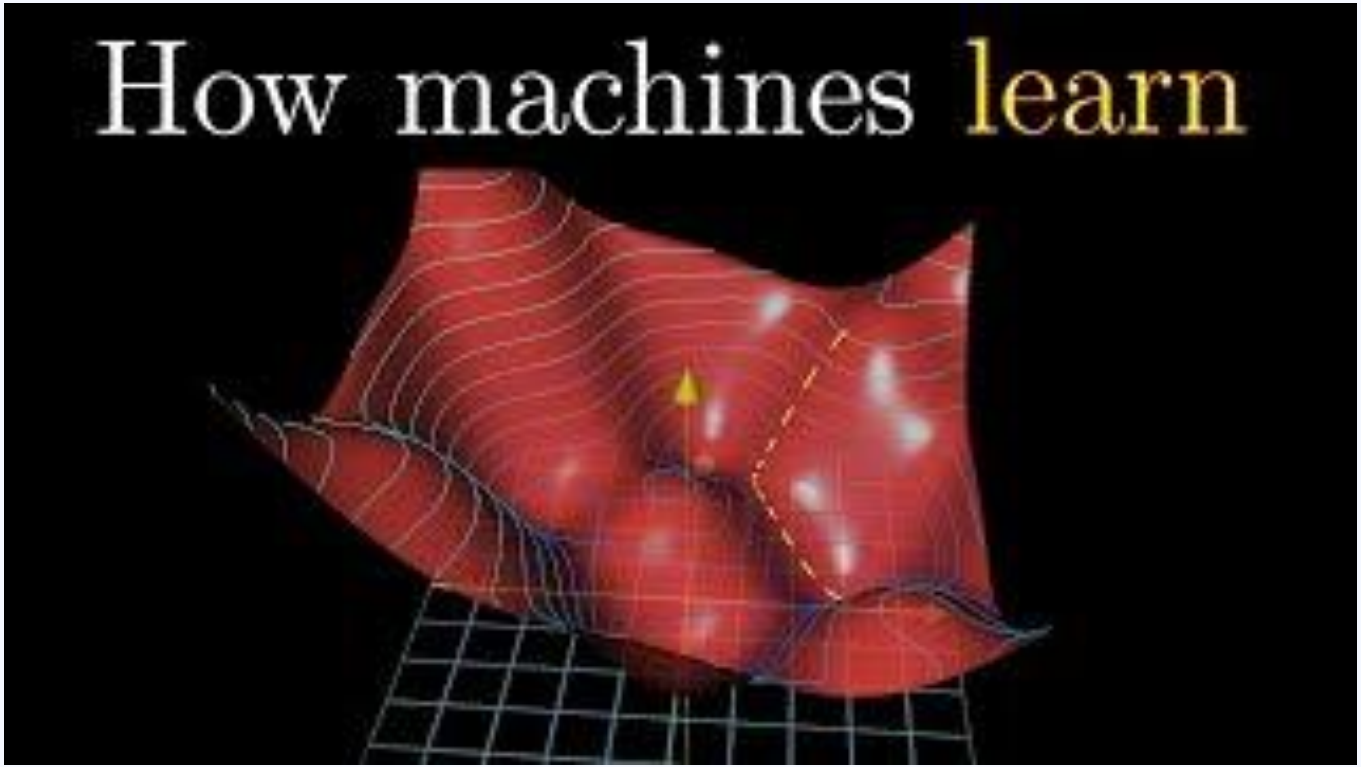
Name	Symbol	Example
Derivative	$\frac{d}{dx}$	$\frac{d}{dx}(x^2) = 2x$
Partial derivative	$\frac{\partial}{\partial x}$	$\frac{\partial}{\partial x}(x^2 - xy) = 2x - y$
Gradient	∇	$\nabla(x^2 - xy) = \begin{bmatrix} 2x - y \\ -x \end{bmatrix}$

Relembrando derivadas

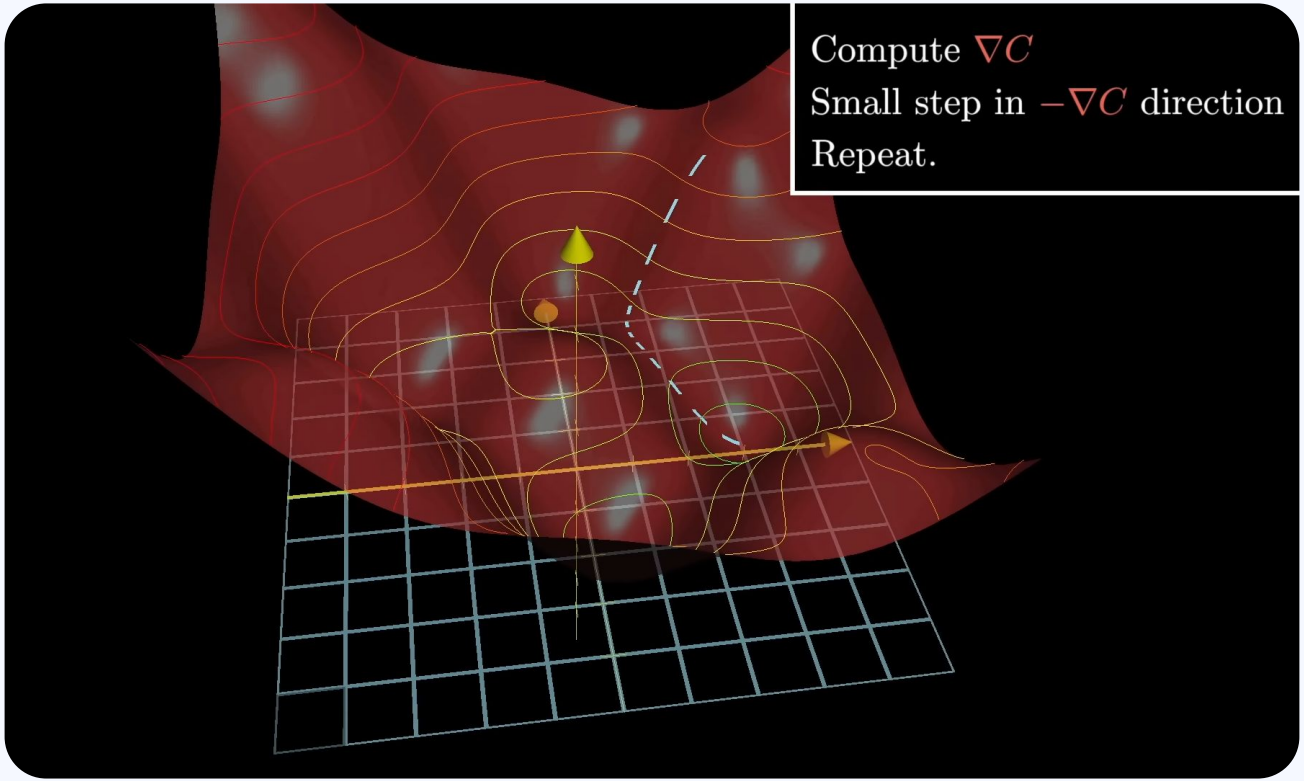



0
−1
−2
−3
−4
−80
−60
−40
−20
0
20
40
60
80
−80
−60
−40
−20
0
20
40
60
80

A descida do gradiente



A descida do gradiente



A descida do gradiente

$$\vec{\mathbf{W}} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{13,000} \\ w_{13,001} \\ w_{13,002} \end{bmatrix}$$

$$-\nabla C(\vec{\mathbf{W}}) = \begin{bmatrix} 0.31 \\ 0.03 \\ -1.25 \\ \vdots \\ 0.78 \\ -0.37 \\ 0.16 \end{bmatrix}$$

w_0 should increase somewhat
w_1 should increase a little
w_2 should decrease a lot

$w_{13,000}$ should increase a lot
$w_{13,001}$ should decrease somewhat
$w_{13,002}$ should increase a little

Treinando uma Regressão Linear

Gradient Descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 0$ and $j = 1$)

}

Linear Regression Model

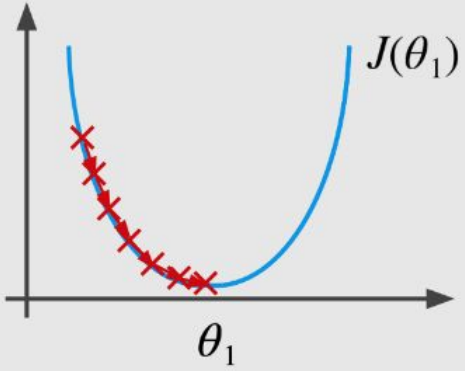
$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

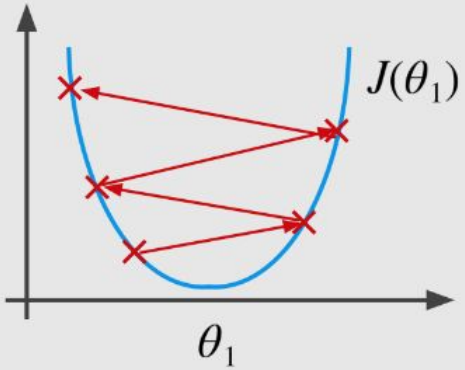
Treinando uma Regressão Linear

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.



If α is too large, gradient descent can be overshoot the minimum. It may fail to converge, or even diverge.



Treinando uma Regressão Linear

$$\frac{\partial}{\partial \theta_j} J(\theta_0,\theta_1) = \frac{\partial}{\partial \theta_j} \cdot \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$= \frac{\partial}{\partial \theta_j} \cdot \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$

$$j = 0: \quad \frac{\partial}{\partial \theta_0} J(\theta_0,\theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$j = 1: \quad \frac{\partial}{\partial \theta_1} J(\theta_0,\theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

Treinando uma Regressão Linear

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x^{(i)}$$

} update θ_0 and θ_1 simultaneously

}

Batch Gradient Descent

Cada **passo**, leva em conta **todas as amostras** de treinamento:

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x^{(i)}$$

} update θ_0 and θ_1 simultaneously

}

Stochastic Gradient Descent

Cada **passo**, leva em conta **uma amostra** de treinamento:

repeat until convergence {

for $i := 1, \ldots, m$ {

$$\theta_0 := \theta_0 - \alpha(h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

}

}

Mini-batch Gradient Descent

Cada **passo**, leva em conta ***b* amostras** de treinamento:

Say $b = 10$, $m = 1000$.

repeat until convergence {

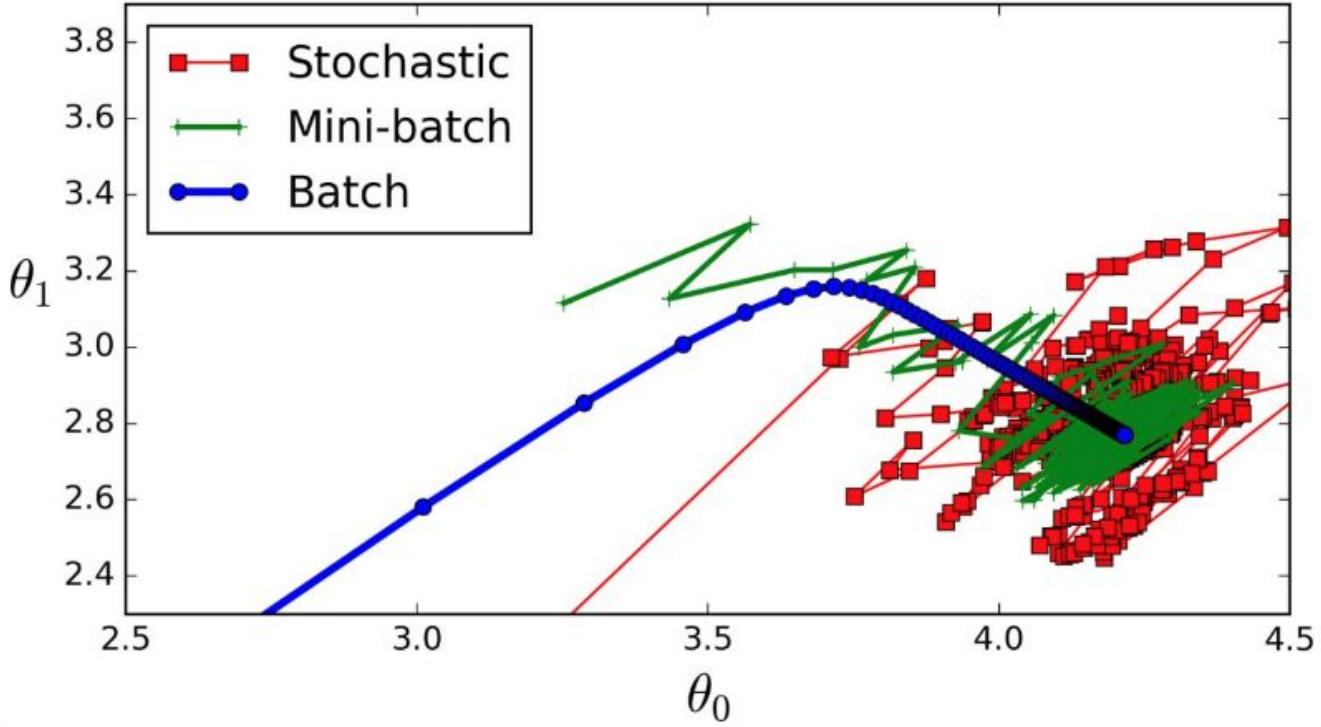
for $i = 1, 11, 21\ldots, 991$ {

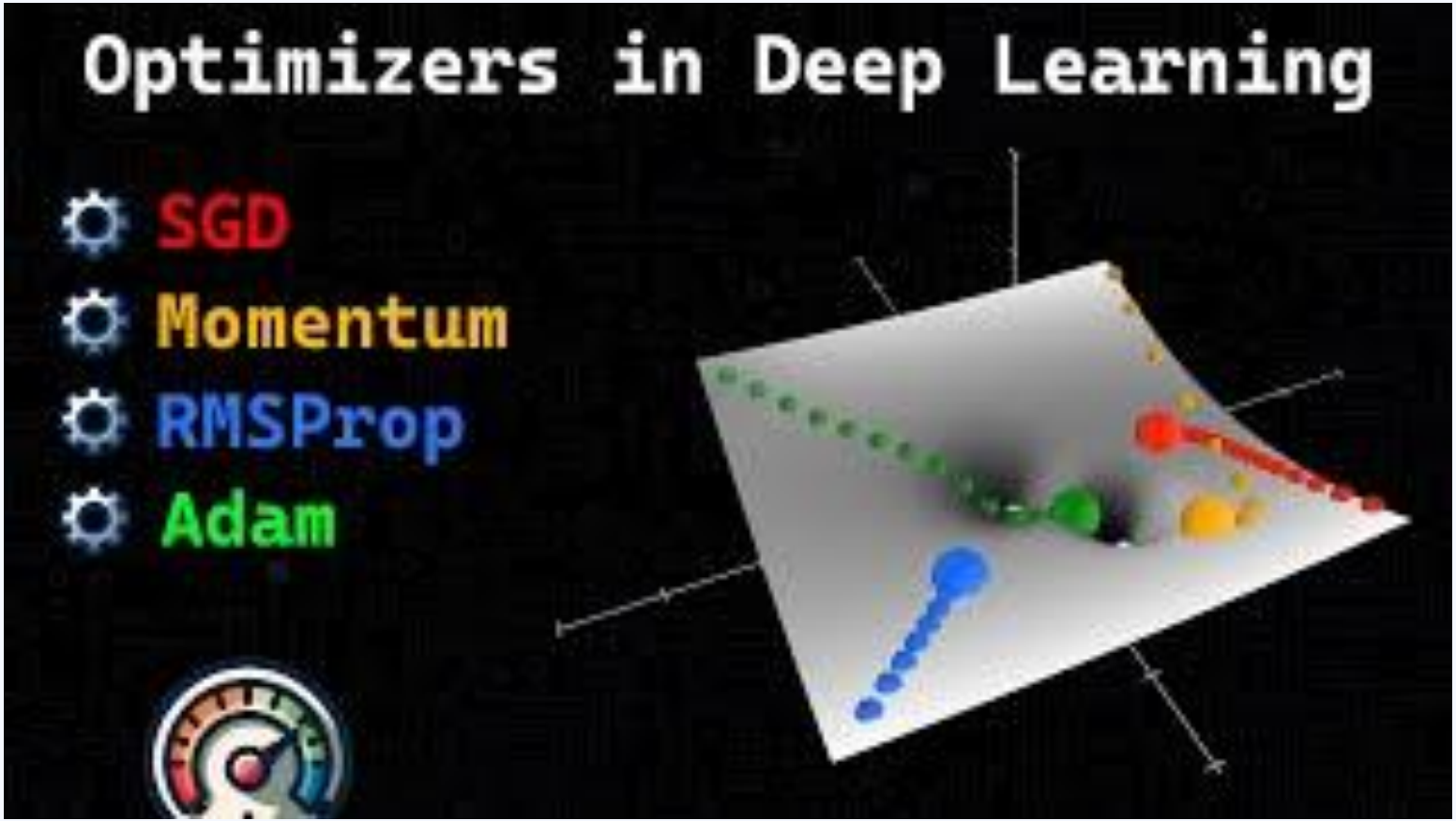
$$\theta_0 := \theta_0 - \alpha \frac{1}{10} \sum_{i=k}^{i+9} (h_\theta(x^{(k)}) - y^{(k)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{10} \sum_{i=k}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x^{(k)}$$

} }

Batch vs Mini-batch vs Stochastic





Instantes: *01:17-04:24; 04:38-04:54; 05:37-07:08; 12:30-12:44; 15:40-16:27; 16:28-16:50.*

ruder.io About Papers Talks FAQ Newsletter NLP Progress Media ••• Sign in Subscribe

optimization

An overview of gradient descent optimization algorithms

Gradient descent is the preferred way to optimize neural networks and many other machine learning algorithms but is often used as a black box. This post explores how many of the most popular gradient-based optimization algorithms such as Momentum, Adagrad, and Adam actually work.

Sebastian Ruder
Jan 19, 2016 • 28 min read

Link para o artigo: https://www.ruder.io/optimizing-gradient-descent/

04

Equação Normal

O que é a Equação Normal?

Basicamente se trata de:

Implementar o Método dos Mínimos Quadrados (MMQ)

Em inglês, *Ordinary Least Squares* (*OLS*)

A **Equação Normal** representa a fórmula para resolver uma Regressão Linear **de maneira analítica**:

$$\frac{d}{d\theta} J(\theta) = \ldots = 0 \quad \text{Solve for } \theta$$

Qual a Equação Normal?

Considerando **m exemplos** e **n *features***, temos:

x_0	Size (feet^2) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($) in 1000's y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$$

Qual a Equação Normal?

Considerando **m exemplos** e **n *features***, temos:

x_0	Size (feet^2) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($) in 1000's y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

Qual a Equação Normal?

Considerando **m exemplos** e **n *features***, temos:

x_0	Size (feet^2) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($) in 1000's y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

Qual a Equação Normal?

Considerando **m exemplos** e **n *features***, temos:

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}_{m \times (n+1)} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}_{m}$$

Qual a Equação Normal?

Considerando **m exemplos** e **n *features***, temos:

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \qquad X = \begin{bmatrix} — & (x^{(1)})^{\mathrm{T}} & — \\ — & (x^{(2)})^{\mathrm{T}} & — \\ — & \vdots & — \\ — & (x^{(m)})^{\mathrm{T}} & — \end{bmatrix}$$

Qual a Equação Normal?

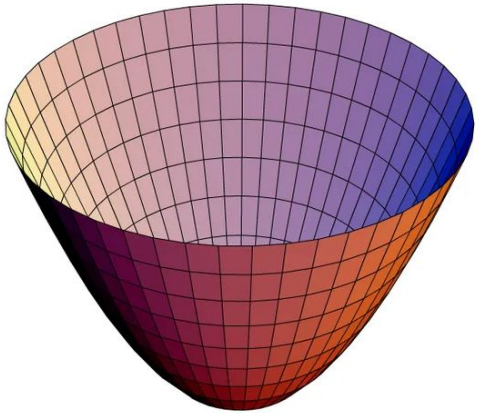
É possível chegar à seguinte expressão:

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

E se a matriz $X^T X$ **não for invertível?**

Provavelmente está utilizando duas *features* altamente correlacionadas, ou seja, **linearmente dependentes**.

Medium Search Write Sign up Sign in



https://qph.ec.quoracdn.net/main-qimg-99f7ea7d0929ed41dbecf67ec51b80b3?convert_to_webp=true

Rohan #3: Deriving the Normal Equation using matrix calculus

Understanding the analytical solution to linear regression.

Rohan Kapur · Follow

Published in A Year of Artificial Intelligence · 11 min read · Feb 16, 2016

1.2K 23

Link para o artigo:
https://ayearofai.com/rohan-3-deriving-the-normal-equation-using-matrix-calculus-1a1b16f65dda

05

Implementando os algoritmos

Obrigado pessoal!

Até próxima aula :)

Iris Data Science UNICAMP

@irisdatascienceunicamp

Referências

Parte do material foi adaptado dos slides da Profª. Sandra Avila apresentados na disciplina MC886.

Material online:

- Gradient descent, how neural networks learn | Chapter 2, Deep learning (YouTube)

Livros utilizados:

- An Introduction to Statistical Learning (2023)
- Hands-On Machine Learning with Scikit-Learn, Keras, and Tensor Flow (2017)