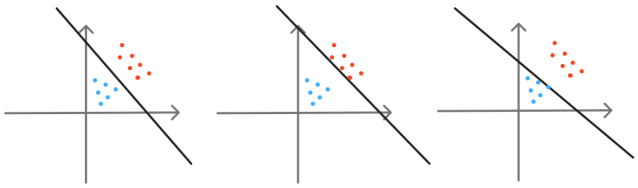
Aula 2

1 Margin Boundary

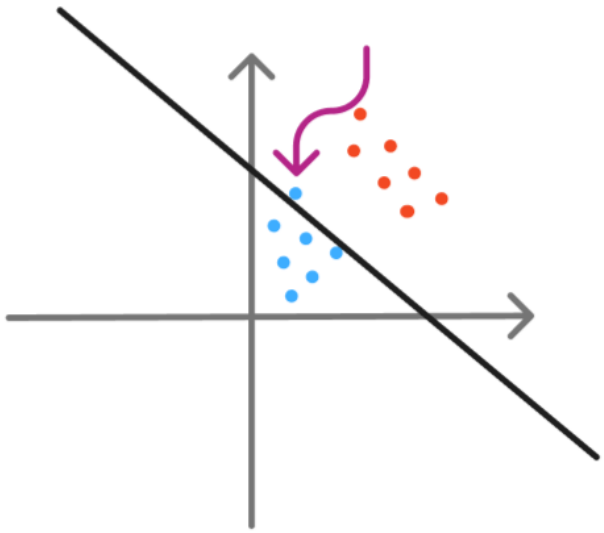
Na última aula introduzimos o algoritmo Perceptron, onde o mesmo era capaz de encontrar uma decision boundary caso o data-set fosse linearmente separável. Porém, será que aquele algoritmo nos da a melhor decision boundary entre todas as possíveis?

Vamos analisar o seguinte exemplo:



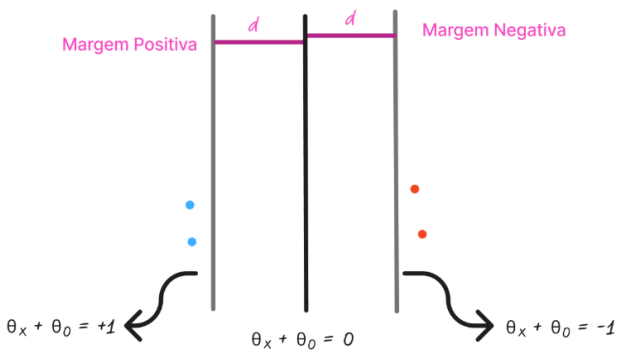
As 3 decision boundary separam corretamente todos os dados, mas qual das 3 é a melhor? Claramente podemos perceber que a melhor entre elas é a da esquerda, pois ela deixa uma margem entre as bolas azuis e as bolas vermelhas.

Caso escolhêssemos a da direita para ser a nossa decision boundary pode ser que o modelo classifique de forma errada um dado azul que ele nunca viu, mesmo que este dado seja extremamente parecido com os dados azuis que ele já viu. Por exemplo:



Isso mostra que na hora de definirmos nossa decision boundary queremos deixar uma certa "margem"entre a mesma e os dados.

A imagem a baixo mostra o exemplo de uma margin boundary:



A margin boundary é composta por duas retas paralelas a decision boundary, margem positiva ($\theta.\vec{x} + \theta_0 = 1$) e margem negativa ($\theta.\vec{x} + \theta_0 = -1$). Ambas as margens estão a mesma distância d da decision boundary.

Na hora de considerar uma decision boundary também devemos levar em consideração a sua margin boundary, porém, qual deve ser a distância que a margin boundary tem que estar da decision boundary?

Sabendo que as margens, tanto positiva quanto negativa são paralelas a decision boundary é possível concluir que a distância d entre a decision boundary e a margin boundary será de:

$$d = \frac{1}{\|\vec{\theta}\|}$$

Ou seja, a distância d dependerá somente do módulo do vetor theta.

A partir disso podemos concluir que achar o valor de θ é equivalente a achar qual é a sua margin boundary. Podemos assim, ao escrever o código, levar em consideração que queremos um valor de θ tal que separe corretamente os dados (igual na última aula) e tenha uma certa margem.

Dizer que queremos ter uma certa margem é abstrato. O que queremos ao final da conta é um valor de θ que gere um d, de tal forma que iremos levar duas coisas em consideração:

1. **Regularização**: favorecer margin boundary o mais longe possível da decision boundary (d alto)
2. **Loss**: quantos exemplos ficaram dentro da margem ao afastá-la da decision boundary.

Queremos favorecer margin boundarys que estão longe da decision boundary já que o objetivo é encontrar uma decision boundary que esteja longe dos pontos azuis e vermelhos, porém ao mesmo tempo devemos levar em consideração quantos pontos ela irá começar a "engolir"ao afastarmos-a.

Sem a loss uma margem infinita seria perfeita, pois ela estaria o mais longe possível da decision boundary, porém isso é inútil.

Função Gamma

Pontos que se encontram dentro da margem são pontos aos quais o modelo não tem muito confiança sobre o resultado. Já os pontos que estão fora da margem estão ou corretamente classificados, ou, classificados de forma errada.

Portanto, agora nosso modelo tem que ser capaz de dizer se o exemplo foi classificado corretamente ou não e se ele está dentro ou fora da margem.

Um ponto ter sido corretamente classificado ou não ainda cai na mesma fórmula da aula passada, isto é:

$$y_i(\vec{\theta}.\vec{x_i} + \theta_0) \leq 0$$

Mas como podemos quantificar se o mesmo está dentro ou fora da margem?

Bom, isso é simples, basta vermos qual a distância daquele ponto a decision boundary. Sabemos que ambas as margens, tanto positiva, quanto negativa estão a uma distância em módulo de $\frac{1}{\|\vec{\theta}\|}$, logo, se um ponto estiver a uma distância, em módulo, maior que $\frac{1}{\|\vec{\theta}\|}$ ele está fora da margem, caso contrário ele está dentro da margem.

Relembrando: A distância entre um ponto (x_1, x_2) e uma reta $(A_1x_1 + A_2x_2 + B = 0)$ é dada por: $\frac{|\ A_1x_1+A_2x_2+B\ |}{\sqrt{A_1+A_2}}$. Fazendo: $\vec{A} = (A_1, A_2)$ e $\vec{x} = (x_1, x_2)$, chegamos que a distância é: $\frac{|\ \vec{A}.\vec{x}+B\ |}{\|\vec{A}\|}$. Logo, a distância de um ponto a decision boundary será de:

$$d = \frac{|\ \vec{\theta}.\vec{x} + \theta_0\ |}{\|\vec{\theta}\|}$$

O numerador, $|\ \vec{\theta}.\vec{x} + \theta_0\ |$, sempre irá retornar um valor positivo, porém, se retirarmos o módulo e multiplicarmo-os por y_i teremos: $y_i(\vec{\theta}.\vec{x_i} + \theta_0)$ que é a função loss da aula passada.

Sendo assim, baseado nessa pequena alteração na equação da distância ficamos com:

$$\gamma(\theta, \theta_0) = \frac{y_i(\vec{\theta}.\vec{x_i} + \theta_0)}{\|\vec{\theta}\|}$$

A função gamma é muito poderosa, pois ela sozinha nos dá todas as informações que precisamos:

- $\gamma(\theta, \theta_0) > 0$, exemplo corretamente classificado
- $\gamma(\theta, \theta_0) < 0$, exemplo classificado errado
- $|\gamma(\theta, \theta_0)| > \frac{1}{\|\vec{\theta}\|}$, exemplo fora da margem
- $0 < |\gamma(\theta, \theta_0)| < \frac{1}{\|\vec{\theta}\|}$, exemplo dentro da margem

Portanto, agora, não vamos mais levar em consideração somente se o modelo está acertando ou errando, porém também iremos levar em consideração o quanto ele está acertando / o quanto ele está errando.

2 Função Loss

Como já foi discutido na aula anterior, ao se falar do algoritmo do perceptron, a função de Loss é responsável por quantificar o erro do nosso modelo, podendo variar dependo do problema.

Na aula passada só nos importávamos se o modelo acertou ou errou, logo, nossa função loss era estritamente binária (1 ou 0). Agora nós sofisticamos o problema, onde, desta vez a distância do ponto a decision boundary é um valor relevante. Caso ocorra um erro, queremos penalizar o modelo proporcionalmente a quão distante aquele ponto está da decision boundary.

2.1 Hinge Loss

Essa função de Loss é utilizada para problemas de classificação, como os que já foram apresentados, sendo definida da seguinte forma:

$$Loss_h(z) = \begin{cases} 0 & \text{if } z \geq 1, \\ 1 - z & \text{if } z < 1. \end{cases}, onde\ z = y_i.(\vec{\theta} \cdot \vec{x_i} + \theta_0)$$

Caso $z \geq 1$ significa que o valor de z é positivo, logo teve um acerto. Caso $z \leq 0$ significa que o valor de z é negativo, logo teve um erro.

Porém, por que verificar que $0 < z < 1$ significa que o exemplo caiu dentro da margem?

Vamos pensar na função γ. Na função gamma, um exemplo está dentro da margem caso:

$$0 < \gamma < \frac{1}{\|\theta\|}$$

O que podemos fazer é o seguinte:

$$0.\|\theta\| < \gamma.\|\theta\| < \frac{1}{\|\theta\|}.\|\theta\|$$

$$0 < \gamma.\|\theta\| < 1$$

Porém: $z = \gamma.\|\theta\|$

Logo:

$$0 < z < 1 = 0 < \gamma < \frac{1}{\|\theta\|}$$

3 Problema da Optimização

A ideia é que queremos fazer com que a Loss seja a menor possível.

$$min(Loss)$$

Porém, também temos que lembrar que na hora que fizemos a introdução sobre a margin boundary dissemos que além da Loss queremos favorecer margin boundarys com uma distância maior a decision boundary. Como a distância da margin boundary até a decision boundary é de $\frac{1}{\|\theta\|}$, queremos:

$$max(\frac{1}{\|\theta\|})$$

Portanto o que queremos é:

$$min(Loss)\ and\ max(\frac{1}{\|\theta\|})$$

Mas, maximizar $\frac{1}{\|\theta\|}$ é equivalente a minimizar $\|\theta\|$, que é equivalente a minimizar $\frac{1}{2}\|\theta\|^2$

Obs: a razão de adicionarmos o 1/2 e o ² é por causa de regra de derivada, simplificando assim a expressão no futuro.

Logo:

$$min(Loss)\ and\ min(\frac{1}{2}\|\theta\|^2)$$

Podemos escrever isso como uma função objetivo da seguinte maneira:

$$J(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^{n} Loss_h + \frac{\lambda}{2}\|\theta\|^2$$

$$J(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^{n} Loss(y_i(\vec{\theta} \cdot \vec{x} + \theta_0)) + \frac{\lambda}{2}\|\theta\|^2$$

A genialidade de minimizar essa função é que minimizar J significa minimizar a loss e minimizar θ, e, minimizar θ significa aumentar a margem.

Assim, nosso objetivo é achar valores de θ e θ_0 que minimizem essa função, pois, assim, estaremos diminuindo o erro do nosso modelo, que é a Loss, e estaremos aumentando as nossas margens, que é inversamente proporcional ao $\|\theta\|$, dando mais segurança e estabilidade para as nossas previsões.

4 Regularização

Este valor de λ, chamado de parâmetro da regularização, é o quanto de importância será dado para a regularização em relação a loss, onde quanto maior for o valor de lambda mais importância está se dando a ter uma margem distante da decision boundary e quanto menor for esse valor mais importância está se dando em errar a menor quantidade de vezes possível.

O termo λ define o "grau de importância"que o aumento do $\|\theta\|$ terá para o aumento de $J(\theta, \theta_0)$.

4.1 Termo de Regularização

Em Machine Learning, nomeamos 'regularização' qualquer técnica que nos permita expandir a capacidade de generalização de nosso modelo, não tornando-o preso aos exemplos de treinamento o que é equivalente a propositalmente reduzir a capacidade do modelo.

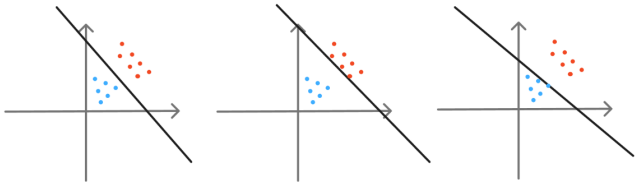
Entrando mais a fundo no Termo de Regularização λ, vemos que o aumento dele acarreta em um maior peso que o $\|\theta\|$ terá no aumento de $J(\theta, \theta_0)$, sendo assim, ao tentarmos minimizar a Função Objetivo, a redução do $\|\theta\|$ terá uma relevância elevada, levando a margens maiores (melhor generalização).

4.2 $\lambda = 0$

Imagina que o seu objetivo é passar em Calculo 1, porém você descobre que a prova do seu professor é igualzinho a lista de exercício. Invés de estudar a matéria você pode simplesmente decorar a lista de exercício e ir fazer a prova.

O que vai acontecer é que você vai tirar 10 na prova, porém você não necessariamente aprendeu a matéria, você somente decorou exercícios para passar nela.

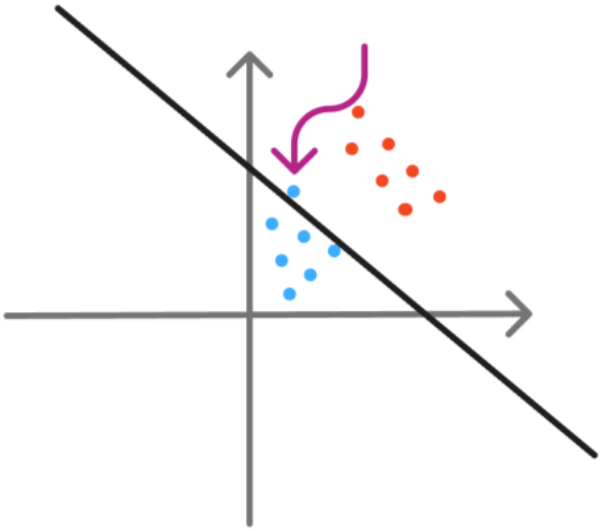
Se não formos cuidadosos o nosso modelo irá fazer o mesmo.



Vamos retornar a imagem do início dessa aula. Você consegue ver como o modelo do meio e o da direta podem simplesmente ter "decorado"os dados.

O modelo foi treinado em um conjunto de dados que chamamos de dados de treinamento, e ele foi dito que o objetivo dele é ter o menor erro possível nos dados de treinamento. Foi o que ele fez, ele conseguiu separar com 100% de acurácia os dados de treinamento.

Mas o que acontece se ele for exposto a um dado que ele nunca viu?



A chance de ele classificar esse dado de forma incorreta é grande, pois ele foi instruído a não errar os dados de treinamento, nada foi dito a ele sobre deixar uma margem para dados que ele nunca viu.

Esse é o tipo de comportamento que temos ao λ ser 0, ou seja, não ter regularização.

Se olharmos a função objetivo com o lambda igual a zero, ela seria:

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} Loss_h(y_i(\vec{\theta} \cdot \vec{x} + \theta_0))$$

Ao minimizarmos essa função estaremos levando em consideração que ela diminua a loss para os dados que ela está vendo, ou seja, os dados de treinamento.

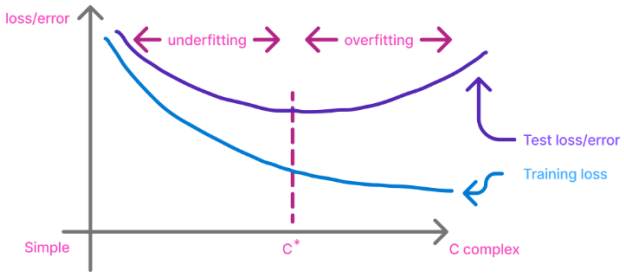
Isso vai fazer com que o modelo overfite, ou seja, ele aprenda a minimizar o máximo o possível a loss no conjunto de testes. Porém, ao fazer isso ele fica ruim em generalizar para dados que ele ainda não viu.

Essa é a importância do λ, a gente precisa deixar uma certa margem para que o modelo não overfite no conjunto de treinamento e seja capaz de generalizar para novos dados.

4.3 Valores de Lambda

Quanto maior for o valor de λ mais importância vai ser dada para ter uma margem grande do que uma loss pequena. Enquanto quanto menor for o valor de λ mais importância vai ser dada para a loss do que ter uma margem grande.

Por questão de simplificação, vamos chamar $c = \frac{1}{\lambda}$. c é a complexidade do modelo, ou seja, quanto maior for o valor de c mais complexo o modelo é, isto, pois, o valor de lambda vai ser pequeno, fazendo com que o modelo se ajuste milimetricamente aos dados (modelo complexo). Já, quanto menor for o valor de c menos complexo vai ser o modelo (valor de lambda grande), exemplificando que a margem do modelo é maior, fazendo com que ele seja mais flexível e assim sendo mais simples.



A imagem a cima demonstra a relação entre o valor de C, o loss do treinamento e o loss do teste. Começando pelo loss do treinamento podemos ver que quanto maior o valor de C menor o loss do treinamento. Entretanto, quando vamos analisar a curva do loss do teste percebemos que o melhor valor para o C, isto é, o $C*$ não é o maior valor de todos, isso acontece pois, conforme passamos desse sweet spot dado por $C*$ acabamos overfitando o modelo, ou seja, ele é muito bom no treinamento, porém ruim no teste, e, antes do sweet spot estamos underfitando o modelo, ou seja, não estamos treinando ele o suficiente, pois ele é ruim no treino e teste.

O valor real de C* não é possível achar, porém é possível encontrar um estimador para ele. Para fazer isso, após o modelo ser treinado no conjunto de treinamento, ele será passado por um conjunto menor de validação e, o valor de C será obtido baseado no valor de C que minimize a loss no conjunto de validação.

5 Derivada

A forma com qual iremos minimizar a nossa função objetivo será através do método de Gradiente Descendente.

Antes de explicar como o Gradiente Descendente funciona precisamos falar o que é derivada.

A derivada é um conceito fundamental do cálculo que representa a taxa de variação instantânea de uma função em um ponto específico. Ou seja, ao calcular a derivada em um ponto específico você será capaz de saber a natureza da atualização daquela função.

Vamos imaginar que a derivada de uma certa função, para $x > 0$ seja: $2x$. Conforme o valor de x aumenta o valor da derivada vai aumentar, isto é, a derivada em 4 é maior que a derivada em 2. Isso significa que conforme o valor de x aumente, o valor da função está aumentando ainda mais.

Para exemplificar a derivada iremos usar a função $f(x) = x^2$

A função $f(x) = x^2$ é uma parábola que tem seu ponto mínimo em $x = 0$, onde $f(0) = 0$. Para qualquer valor de x diferente de zero, o valor da função é positivo e cresce à medida que nos afastamos da origem.

A derivada de $f(x) = x^2$

A derivada desta função é $f'(x) = 2x$.
O que isso significa intuitivamente? Para cada ponto x:

- Em $x = 1$, a derivada é $f'(1) = 2$, indicando que a função está crescendo com inclinação 2
- Em $x = 2$, a derivada é $f'(2) = 4$, mostrando que a função cresce ainda mais rapidamente
- Em $x = 0$, a derivada é $f'(0) = 0$, significando que a função não está crescendo nem diminuindo (ponto mínimo)
- Em $x = -1$, a derivada é $f'(-1) = -2$, indicando que a função está diminuindo

Andar na direção oposta da derivada

Agora, vamos entender o que significa "andar na direção oposta da derivada":

1. Para $x > 0$ (lado direito do gráfico):
    - A derivada $f'(x) = 2x$ é positiva
    - A direção oposta seria negativa
    - Mover-se para a esquerda (em direção ao zero) faz o valor da função diminuir
2. Para $x < 0$ (lado esquerdo do gráfico):
    - A derivada $f'(x) = 2x$ é negativa
    - A direção oposta seria positiva
    - Mover-se para a direita (em direção ao zero) faz o valor da função diminuir
3. No ponto $x = 0$:
    - A derivada $f'(0) = 0$ é zero
    - Não há direção clara a seguir
    - Estamos no ponto mínimo da função, onde seu valor é o menor possível

Este é o princípio básico da "descida do gradiente", um método amplamente utilizado em otimização e aprendizado de máquina. Ao seguir o caminho oposto da derivada, sempre nos movemos na direção que faz o valor da função diminuir.

Sendo assim, para diminuir o valor da função objetivo devemos sempre andar na direção oposta da derivada.

Só existe um problema. O conceito apresentado a cima mostra como que eu faço para diminuir o valor de uma função que só tem uma única variável, que neste caso é o x, porém nossa função objetivo contém duas variáveis, o θ e θ_0, além disso, θ é um vetor.

Para resolver este tipo de problema, onde a função tem uma dimensionalidade maior do que 1 precisamos introduzir agora a noção de gradiente.

6 Gradient

Agora, se tivermos uma função de mais de uma variável? A ideia é que para esses casos a gente quer saber como que cada uma das variáveis influencia na função individualmente.

Por exemplo, se temos uma função que contém duas variáveis, x e y: $F(x, y)$. O que devemos fazer é calcular o vetor gradiente desta função, que será dado por:

$$\vec{\nabla}F(x, y) = (\frac{\partial F(x, y)}{\partial x}, \frac{\partial F(x, y)}{\partial y})$$

$\frac{\partial F(x,y)}{\partial x}$ é o que chamamos de derivada parcial da função F em relação a x. Essa equação é uma equação de derivada, e o que ela faz é. Ela assume que y é uma constante, não uma variável e calcula a derivada em relação a x. Sendo assim, o resultado desta expressão dará qual é a taxa de atualização da função $F(x, y)$ em relação a x, em outras palavras, qual a influência de x na função $F(x, y)$ naquele exato ponto.

Analogamente, $\frac{\partial F(x,y)}{\partial y}$ é a derivada parcial de $F(x, y)$ em relação a y.

Dessa forma, o vetor $\vec{\nabla}F$ irá nos dizer em qual sentido nossa função está indo, com relação a cada uma das suas variáveis.

Como queremos minimizar nossa função objetivo J o que queremos fazer é andar na direção oposta ao gradiente.

7 Learning Rate

Antes de irmos para o algoritmo em si precisamos introduzir um novo conceito chamado de learning rate.

Vamos pensar na função $f(x) = x$. Imagine que o meu x atual é 4 e eu gostaria de minimizar o máximo possível esta função.

Como já vimos, a sua derivada, $f'(x)$ é $2x$. Logo, no ponto $x = 4$, teremos que sua derivada é: $f'(4) = 8$.

Pelo o que estudamos devemos atualizar o valor de x andando na direção contrária da derivada. Logo:

$$x_{novo} = x_{antigo} - f'(x)$$

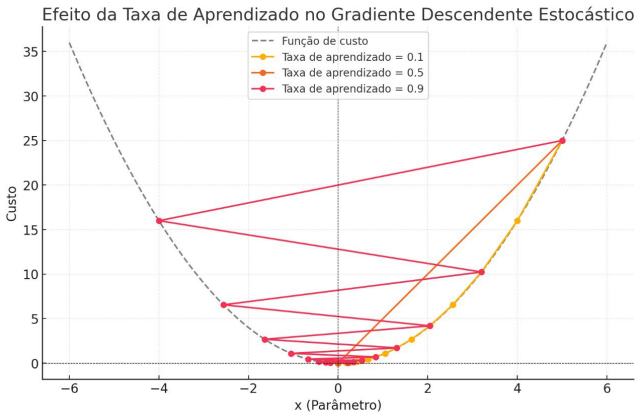
$$x = 4 - 8; x = -4$$

Agora vamos aplicar o passo novamente para $x = -4$:

$$x = -4 - (-4 * 2); x = -4 - (-8); x = -4 + 8; x = 4$$

Como você pode perceber caímos em um loop. Isso acontece, pois, as mudanças que estamos fazendo na atualização são muito drásticas, e como já estamos muito perto do menor valor possível, isto é, $x = 0$, ficaremos presos para sempre entre $x = 4$ e $x = -4$.

Para resolvermos este problema devemos utilizar uma variável chamada Learning Rate η. Ela diz quanto do real valor do gradiente iremos utilizar para atualizar a nossa variável.



Por exemplo, se $\eta = 0.1$, na hora de atualizar x faremos:

$$x_{novo} = x - 0,1.f'(x)$$

A ideia é que iremos começar com uma learning rate alta e a cada iteração iremos diminuí-la.

8 Algoritmo utilizando Gradiente

A ideia é que iremos iterar sob o conjunto de dados T vezes. Cada vez iremos pegar um elemento aleatório do conjunto de dados e andar na direção oposta do gradiente para θ e θ_0:

1. θ = valor aleatório; θ_0 = valor aleatório
2. for t in range(T)
3. sortear i em: $\{x_0, x_1, \ldots, x_n\}$ (1)
4. $\theta = \theta - \eta\nabla_\theta[J]$
5. $\theta_0 = \theta_0 - \eta\nabla_{\theta_0}[J]$

Calculando os gradientes:

$$\nabla_\theta J = \begin{cases} 0, \text{if loss } 0, \\ -y_i\vec{x_i}, \text{if loss} < 0 \end{cases} + \lambda\theta$$

$$\nabla_{\theta_0} J = \begin{cases} 0, \text{if loss } 0, \\ -y_i, \text{if loss} < 0 \end{cases}$$

Vamos expandir a conta do gradiente descendente:

$$\theta = \theta - \eta\nabla_\theta[J]$$

$$\theta_0 = \theta_0 - \eta\nabla_{\theta_0}[J]$$

Caso Loss < 0:

$$\theta = \theta - \eta.(-\vec{x_i}y_i + \lambda\theta)$$

$$\theta = \theta + \eta.(\vec{x_i}y_i) - \eta(\lambda\theta)$$

$$\theta_0 = \theta_0 - \eta.(-y_i)$$
$$\theta_0 = \theta_0 + \eta.y_i$$

Com loss = 0:

$$\theta = \theta - \eta.(\lambda\theta)$$

$$\theta_0 = \theta_0 - \eta.0$$

Sendo assim, o algoritmo será:

1. $\theta =$ valor aleatório; $\theta_0 =$ valor aleatório
2. for t in range(T)
3. sortear i em: $\{\vec{x_0}, \vec{x_1}, \ldots, \vec{x_n}\}$
4. if Loss ≤ 0 :
4. $\theta = \theta + \eta.(\vec{x_i}y_i) - \eta(\lambda\theta)$
5. $\theta_0 = \theta_0 + \eta.y_i$
6. $else$:
7. $\theta = \theta - \eta(\lambda\theta)$

(2)