Slide 1

TP555 - Inteligência Artificial e Machine Learning:
Regressão Linear

Inatel

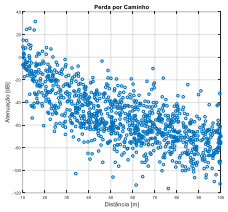
Felipe Augusto Pereira de Figueiredo
felipe.figueiredo@inatel.br

A essa altura, espero que vocês consigam diferenciar problemas de regressão e classificação.

Slide 2

Motivação

- **Exemplo 1**: Predição da perda por caminho (do inglês, *path-loss*).
- **Exemplo 2**: Estimação do preço de casas.



R$ 1.000.000,00

R$ 200.000,00

???

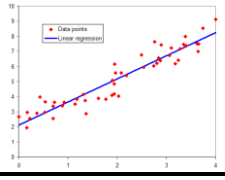
- Podemos encontrar uma relação matemática entre a distância e a atenuação?
- Ou entre a área de uma casa e seu valor?

Modelos para predição da perda por caminho.
Modelos para predição do preço de casas.
Modelos para predição do salário de um empregado baseado em sua idade.

Slide 3

Motivação

- Um dos mais, se não o mais, comum/conhecido algoritmo de aprendizado de máquina.
- Vai nos dar vários *insights (*i.e., intuições*)* importantes para o entendimento de outros algoritmos mais complexos.
- **Regressão**: termo cunhado no século 19 para descrever fenômeno onde altura de descendentes de ancestrais altos tendem a *regredir* para a média.
- **Objetivo**: encontrar uma função, h, que mapeie as variáveis de entrada x em uma variável de saída y: $y = h(x)$.



insight: percepção; intuição; compreensão.

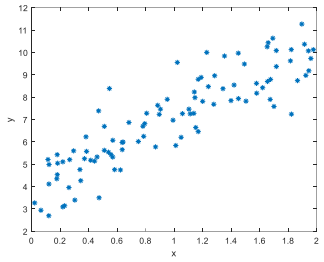
O termo "regressão" foi cunhado por Francis Galton no século 19 para descrever um fenômeno biológico. O fenômeno foi que as alturas dos descendentes de ancestrais altos tendem a regredir para a média (um fenômeno também é conhecido como regressão à média)

Os algoritmos de aprendizado de máquina são descritos como o aprendizado de uma função, *h*, que melhor mapeie as variáveis de entrada x em uma variável de saída y: y = *h*(x)

Slide 4

Regressão Linear

- Dado que você tenha entradas (features/exemplos), ***x***, e saídas (labels), ***y***, de um sistema modelado por uma ***função objetivo*** desconhecida, $f(\boldsymbol{x})$.
- O que você faria para encontrar uma função, $h(\boldsymbol{x})$, que ***aproximasse*** $f(\boldsymbol{x})$?



Por quê Regressão LINEAR?
A regressão linear é chamada linear porque você modela sua variável de saída (y) como uma **combinação linear** de entradas e pesos (vamos chamá-las de x e a, respectivamente).
Linear significa "linear com relação aos parâmetros" e não às variáveis, i.e., x. Portanto os seguintes modelos também são lineares com relação aos parâmetros.

- y = a0 + a1*log(x1) + a2*cos(x2)
- y = a0 + a1*e^x1
- y = a0 + a1*x1^2

Regressão também é conhecida como **aproximação de funções**.

Basicamente, regressão significa encontrar a melhor curva para seus dados numéricos, ou seja, uma aproximação funcional dos dados.

O ***espaço de hipóteses*** é o conjunto de todas a possíveis ***funções hipótese***: h(x) a_0 + a_1 x_1 + a_2 x_2 + ... + a_k x_k. Esse conjunto de todas as possíveis funções forma um hiperplano.
A escolha do espaço de hipóteses é importantíssima para a complexidade da tarefa de encontrar uma boa hipótese para a função desconhecida, ***f***. Um espaço de hipóteses muito grande faz com que se leve muito tempo para se ecntrar ***h***.

Slide 5

Regressão Linear

- Na literatura, $h(\boldsymbol{x})$, é chamada de ***função hipótese*** pois é uma das possíveis soluções encontradas no ***espaço de hipóteses***,

$$\boldsymbol{H} \rightarrow h(\boldsymbol{x}) = a_0 + a_1 x_1 + \cdots + a_K x_K = a_0 + \sum_{i=1}^{K} a_i \, x_i.$$

- **Espaço de hipóteses**: conjunto de todas as possíveis ***funções hipótese***.
  - Hiperplano formado por todos possíveis valores dos parâmetros, $a_k, \forall k$.
- **Objetivo**: Encontrar os parâmetros a_0, a_1, …, a_k de tal forma que $h(\boldsymbol{x})$ seja uma ótima aproximação de $f(\boldsymbol{x})$.
- **Aprendizado supervisionado**: atributos/exemplos (i.e., ***x***) + rótulos/objetivos (i.e., ***y***).

Por quê Regressão LINEAR?
A regressão linear é chamada linear porque você modela sua variável de saída (y) como uma **combinação linear** de entradas e pesos (vamos chamá-las de x e a, respectivamente).
Linear significa "linear com relação aos parâmetros“ e não às variáveis, i.e., x. Portanto os seguintes modelos também são lineares com relação aos parâmetros.

- y = a0 + a1*log(x1) + a2*cos(x2)
- y = a0 + a1*e^x1
- y = a0 + a1*x1^2

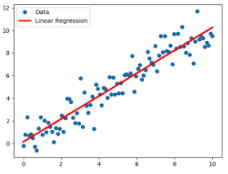
Regressão também é conhecida como **aproximação de funções**.

Basicamente, regressão significa encontrar a melhor curva para seus dados numéricos, ou seja, uma aproximação funcional dos dados.

O ***espaço de hipóteses*** é o conjunto de todas a possíveis ***funções hipótese***: h(x) a_0 + a_1 x_1 + a_2 x_2 + ... + a_k x_k. Esse conjunto de todas possíveis funções forma um hiperplano.
A escolha do espaço de hipóteses é importantíssima para a complexidade da tarefa de encontrar uma boa hipótese para a função desconhecida, ***f***. Um espaço de hipóteses muito grande faz com que se leve muito tempo para se econtrar ***h***.

Slide 6

Definição do Problema



O problema de regressão pode ser enunciado da seguinte forma:

- **Dados disponíveis (ou observações)**: $\{\boldsymbol{x}(i), y(i)\}$, $i = 0, \dots, N-1$, onde
  - $\boldsymbol{x}(i) \in \mathbb{R}^K$: i-ésimo vetor de entrada com dimensão K, ou sejam K atributos (ou features).
  - $y(i) \in \mathbb{R}$: i-ésimo valor esperado de saída referente ao vetor de entrada $\boldsymbol{x}(i)$.
- **Modelo**:

$$h(\boldsymbol{x}(i)) = a_0 + a_1 x_1(i) + \cdots + a_K x_K(i) = \boldsymbol{a}^T \Phi(i),$$

onde $\boldsymbol{a} = [a_0, \dots, a_K]^T$ e $\Phi(i) = [1, x_1(i), \dots, x_K(i)]^T$.

  - $\boldsymbol{a}$ é o vetor $(K+1 \times 1)$ contendo os parâmetros/pesos que definem a ***função hipótese***, ou seja, o mapeamento h: $\boldsymbol{x}(i) \rightarrow \hat{y}(i)$ e $\Phi(i)$ é um vetor $(K+1 \times 1)$ contendo os i-ésimo exemplos.
- **Objetivo do modelo**: encontrar o vetor $\boldsymbol{a}$ que minimize o ***erro***, dado por uma ***função de erro***, $J_e(\boldsymbol{a})$, entre a aproximação $\hat{y}(i)$ e o valor desejado $y(i)$ para todo i.

$$\min_{\boldsymbol{a}} J_e(\boldsymbol{a})$$

Ou seja, **o treinamento do modelo envolve a minimização de uma *função de erro*.**

A função de erro também é conhecida como função de perda ou função de custo.

Note que o vetor x inclui uma entrada fixa de valor unitário que está relacionada ao coeficiente, conhecido como bias ou termo de polarização

Slide 7

Função de Erro

Função de erro: existem várias possibilidades para se definir a função de erro a ser minimizada, porém, geralmente, utiliza-se a medida do **erro quadrático médio**

$$J_e(\boldsymbol{a}) = \frac{1}{N}\sum_{i=0}^{N-1}(y(i) - \hat{y}(i))^2 = \frac{1}{N}\sum_{i=0}^{N-1}\big(y(i) - h(\boldsymbol{x}(i), \boldsymbol{a})\big)^2.$$

A função erro pode ser reescrita em forma matricial como

$$J_e(\boldsymbol{a}) = \frac{1}{N}\|\boldsymbol{y} - \boldsymbol{\Phi a}\|^2 = \frac{1}{N}\|\boldsymbol{e}\|^2 = \frac{1}{N}(\boldsymbol{y} - \boldsymbol{\Phi a})^T(\boldsymbol{y} - \boldsymbol{\Phi a}),$$

onde $\boldsymbol{y} = [y(0), \dots, y(N-1)]^T$ é um vetor $(N \times 1)$, $\boldsymbol{\Phi} = [\Phi(0), \dots, \Phi(N-1)]^T$ é uma matriz $(N \times K + 1)$ e N é o número de amostras ou observações.

Então, para encontrar o vetor de parâmetros $\boldsymbol{a}$ devemos minimizar

$$\min_{\boldsymbol{a}\in\mathbb{R}}\|\boldsymbol{y} - \boldsymbol{\Phi a}\|^2 = \min_{\boldsymbol{a}\in\mathbb{R}}\|\boldsymbol{e}\|^2.$$

O uso do **erro quadrático médio** remonta a Gauss, que demonstrou que quando os valores de y(i) tem ruído com distribuição normal, então os valores mais prováveis para o vetor de parâmetros/pesos **a** são obtidos através da minimização da soma dos quadrados dos erros.

Slide 8

Minimizando a Função de Erro

Expandindo $\|\boldsymbol{e}\|^2 = \|\boldsymbol{y} - \boldsymbol{\Phi a}\|^2$ temos

$$\|\boldsymbol{e}\|^2 = (\boldsymbol{y} - \boldsymbol{\Phi a})^T(\boldsymbol{y} - \boldsymbol{\Phi a}) = \boldsymbol{y}^T\,\boldsymbol{y} - \boldsymbol{y}^T\,\boldsymbol{\Phi a} - \boldsymbol{a}^T\boldsymbol{\Phi}^T\,\boldsymbol{y} + \boldsymbol{a}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi a}$$

Como encontramos o mínimo dessa função em relação aos parâmetros?

Sabemos que derivando $\|\boldsymbol{e}\|^2$ com relação a $\boldsymbol{a}$ e igualando a 0 nós encontramos o ponto onde a inclinação da função de erro é nula,

$$\frac{\partial\|\boldsymbol{e}\|^2}{\partial\boldsymbol{a}} = 2\boldsymbol{a}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi} - 2\boldsymbol{y}^T\,\boldsymbol{\Phi} = 0,$$

porém, esse ponto pode ser tanto um mínimo como um máximo.

Como sabemos se o ponto encontrado é um mínimo ou um máximo?

Se a inclinação é nula e a derivada de segunda ordem for positiva, então o ponto nós dá o mínimo,

$$\frac{\partial^2\|\boldsymbol{e}\|^2}{\partial^2\boldsymbol{a}} = 2\boldsymbol{\Phi}^T\boldsymbol{\Phi}.$$

Se a matriz $\boldsymbol{\Phi}$ tiver ***posto*** igual a $K+1$, então a matriz $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ é ***positiva semi-definida*** e portanto o ponto encontrado acima é realmente o mínimo da ***função de erro***.

O mínimo é encontrado através do método dos mínimos quadrados.

Uma derivada basicamente encontra a inclinação de uma função. A inclinação é nula nos mínimos e máximos de uma função. Se a inclinação é nula e a segunda derivada é menor do que zero, então este é um máximo local, por outro lado, se a derivada segunda for positiva, então, este é um mínimo local.

Posto de uma matriz: corresponde ao número de linhas ou colunas linearmente independentes da matriz.

Se A é uma matriz quadrada (ou seja, m = n), então A é invertível se e somente se A tiver posto igual a n (ou seja, A tiver posto completo).

O método dos mínimos quadrados só funciona para sistemas sobredeterminados, ou seja, conjuntos de equações nas quais há mais equações (pares x e y) que incógnitas (valores do vetor a).

Slide 9

Minimizando a Função de Erro

Portanto, voltando à eq. da derivada primeira igual a 0, temos

$$\boldsymbol{a}^T \boldsymbol{\Phi}^T \boldsymbol{\Phi} = \boldsymbol{y}^T \boldsymbol{\Phi}.$$

Após aplicarmos o transposto a ambos os lados e isolando $\boldsymbol{a}$ temos

$$\boldsymbol{a} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{y}.$$

Essa equação é conhecida como a ***equação normal***.

OBS.1: O método encontra uma ***solução única*** se e somente se a matriz quadrada $\boldsymbol{\Phi}^T \boldsymbol{\Phi}$ for invertível (ou não-singular), ou seja, com ***posto*** igual a $K + 1$.

OBS.2: O método só funciona para sistemas ***sobredeterminados***, ou seja, mais equações (exemplos) do que incógnitas (features), $N \geq K + 1$.

OBS.3: Para sistemas ***subdeterminados*** a matriz $\boldsymbol{\Phi}^T \boldsymbol{\Phi}$ tem posto menor do que $K + 1$ e portanto é ***singular***. Não existe solução ou ela não é única.

O mínimo é encontrado através do método dos mínimos quadrados.

Uma derivada basicamente encontra a inclinação de uma função. A inclinação é nula nos mínimos e máximos de uma função. Se a inclinação é nula e a segunda derivada é menor do que zero, então este é um máximo local, por outro lado, se a derivada segunda for positiva, então, este é um mínimo local.

Se A é uma matriz quadrada (ou seja, m = n), então A é invertível se e somente se A tiver posto igual a n (ou seja, A tiver posto completo).

O método dos mínimos quadrados só funciona para sistemas sobredeterminados, ou seja, conjuntos de equações nas quais há mais equações (pares x e y) que incógnitas (valores do vetor a).

Sistemas subdeterminados (número de exemplos menor do que o número de parâmetros) não tem solução ou não tem solução única.

Slide 10

Matriz Singular

- Uma matriz é ***singular*** quando um ou mais de seus ***autovalores*** são iguais a 0.
- Geometricamente, um ***autovalor*** igual a 0 significa inexistência de informação na direção do ***autovetor*** correspondente.
- Portanto, a matriz não tem ***posto completo*** o que significa que as colunas não são ***linearmente independentes***, ou seja, uma ou mais colunas são versões escalonadas das outras.
- O determinante de uma matriz quadrada $\boldsymbol{A}$ é igual ao produto de seus ***autovalores***

$$\det(\boldsymbol{A}) = \prod_i \lambda_i$$

- Dada a matriz $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a} & \boldsymbol{b} \\ \boldsymbol{c} & \boldsymbol{d} \end{bmatrix}$, caso ao menos um de seus ***autovalores*** seja igual a zero, temos

$$\boldsymbol{A^{-1}} = \frac{1}{\det(\boldsymbol{A})} \begin{bmatrix} \boldsymbol{d} & -\boldsymbol{b} \\ -\boldsymbol{c} & \boldsymbol{a} \end{bmatrix},$$

onde $\frac{1}{\det(\boldsymbol{A})} = \infty$ e portanto, $\boldsymbol{A}$ não é invertível.

- **Solução:** ***regularização de matrizes***, a qual adiciona restrições ao problema da minimização de tal forma que a matriz a ser invertida deixa de ser singular.

O determinante de uma matriz é igual ao produto de todos os seus autovalores. Portanto, se um ou mais autovalores forem iguais a zero, o determinante é zero e, portanto, essa é uma matriz singular, ou seja, não-invertível.
Geometricamente, um autovalor igual a zero significa nenhuma informação na direção do autovetor correspondente. Portanto, você pode reduzir a dimensão sem perder nenhuma informação. No entanto, nos algoritmos de redução de dimensão (e.g., PCA), na verdade perdemos alguma informação.

O **posto** de uma matriz é dado pelo número de ***autovalores*** diferentes de 0. Apenas matrizes com posto completo possuem inversas.

Quantos planos, 2D, podem passar por uma reta, 1D?
Resposta: Infinitos. A solução pra esse sistema de equações é infinito e dada pela reta onde todos os planos se cruzam.

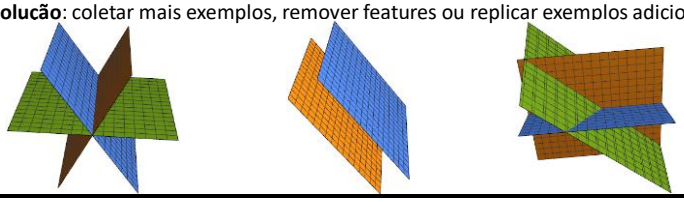
Técnicas de seleção de features retém apenas um subconjunto das features mais relevantes, segundo algum critério, para o problema.
Exemplo de técnica: LASSO (least absolute shrinkage and selection operator) realiza seleção e regularização de variáveis para aprimorar a precisão e a interpretabilidade das previsões do modelo estatístico que produz. O método constrói um modelo linear, que penaliza os coeficientes de regressão (vetor $\boldsymbol{a}$) com uma penalidade L1, reduzindo muitos deles a zero. Features que tem o coeficiente de regressão igual a zero são eliminadas.

More-Penrose inverse, ou pseudo-inversa

Slide 11

Causas para singularidade de $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$

- Features, $\boldsymbol{x}$, redundantes: pelo menos um das features $x_1, x_2, \dots, x_K$ é linearmente dependente.
  - **Ex.**: x_1 = área em m^2 e x_2= área em $pés^2$ ⇒ 1 m^2 = 10.7639 $pés^2$.
  - **Solução**: remover uma das features que são linearmente dependentes.
    - ❑**Técnicas**: seleção de features (LASSO) e redução de dimensionalidade (PCA).
- Número de features ($x_1, x_2, \dots, x_K$) maior do que o de observações ($y(0), y(1), \dots, y(N-1)$), i.e., $K > N$ (***sistema subdeterminado***):
  - ***Sistema subdeterminado***: sistema não tem solução ou existe um número infinito delas (Ex.: quantos planos, 2D, podem passar por uma reta, 1D?).
  - **Solução**: coletar mais exemplos, remover features ou replicar exemplos adicionando ruído.



O determinante de uma matriz é igual aos produtos de todos os seus autovalores. Portanto, se um ou mais autovalores forem iguais a zero, o determinante é zero e, portanto, essa é uma matriz singular, ou seja, não-invertível.
Geometricamente, autovalor igual a zero significa nenhuma informação em um eixo. Portanto, você pode reduzir a dimensão sem perder nenhuma informação. No entanto, nos algoritmos de redução de dimensão (e.g., PCA), na verdade perdemos alguma informação.

Quantos planos, 2D, podem passar por uma reta, 1D?
Resposta: Infinitos. A solução pra esse sistema de equações é infinito e dada pela reta onde todos os planos se cruzam.

Para um sistema que envolve duas variáveis (x e y), cada equação linear determina uma linha no plano xy. Como uma solução para um sistema linear deve satisfazer todas as equações, o conjunto de soluções é a interseção dessas linhas e, portanto, é uma linha, um ponto único ou o conjunto vazio. Para três variáveis, cada equação linear determina um plano no espaço tridimensional e o conjunto de soluções é a interseção desses planos. Assim, o conjunto de soluções pode ser um plano, uma linha, um ponto único ou o conjunto vazio. Por exemplo, como três planos paralelos não têm um ponto comum, o conjunto de soluções de suas equações está vazio; o conjunto de soluções das equações de três planos que se cruzam em um ponto é ponto único; se três planos passam por dois pontos, suas equações têm pelo menos duas soluções comuns; de fato, o conjunto de soluções é infinito e consiste em toda a linha que passa por esses pontos.

Técnicas de seleção de features retém apenas um subconjunto das features mais relevantes, segundo algum critério, para o problema.
Exemplo de técnica: LASSO (least absolute shrinkage and selection operator) realiza seleção e regularização de variáveis para aprimorar a precisão e a interpretabilidade das previsões do

modelo estatístico que produz. O método constrói um modelo linear, que penaliza os coeficientes de regressão (vetor ***a***) com uma penalidade L1, reduzindo muitos deles a zero. Features que tem o coeficiente de regressão igual a zero são eliminadas.

More-Penrose inverse, ou pseudo-inversa

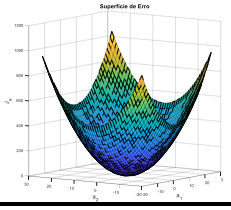
Slide 12

Superfície de Erro

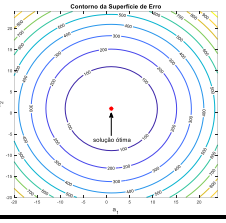
E se plotarmos a função de erro, $J_e(\boldsymbol{a})$?

$$J_e(\boldsymbol{a}) = \boldsymbol{y}^T\,\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{\Phi}\boldsymbol{a}^T - \boldsymbol{a}^T\boldsymbol{\Phi}^T\,\boldsymbol{y} + \boldsymbol{a}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{a}.$$

- $J_e(\boldsymbol{a})$ faz o mapeamento entre cada possível valor dos parâmetros do modelo e o erro correspondente: $J_e(\boldsymbol{a})$: $\mathbb{R}^{K+1} \rightarrow \mathbb{R}$. Esse mapeamento define a ***superfície de erro***.
- Percebam que $J_e(\boldsymbol{a})$ assume uma forma ***quadrática*** com respeito ao ***vetor de parâmetros***, $\boldsymbol{a}$.
- Consequentemente é ***convexa*** (forma de tigela), e portanto possui um único ***mínimo global***, que pode ser encontrado, por exemplo, pela ***equação normal***.
- Como $\frac{\partial^2 \|J_e(\boldsymbol{a})\|^2}{\partial^2 \boldsymbol{a}} = 2\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ é uma matriz positiva semi-definida, $J_e(\boldsymbol{a})$ é sempre convexa com relação a $\boldsymbol{a}$.
- $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ é uma matriz positiva semi-definida pois $\boldsymbol{x}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{x} = \|\boldsymbol{\Phi}\boldsymbol{x}\|^2$.



$y(n) = x_1(n) + x_2(n) + w(n)$,
onde $x_1(n), x_2(n)$, e $w(n) \sim N(0,1)$
e $a_1 = a_2 = 1$.



No problema de regressão linear, a superfície de erro para o critério de quadrados mínimos é dada pela expressão: $\partial J_e(\boldsymbol{a}) = \boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{\Phi}\boldsymbol{a} - \boldsymbol{a}^T\boldsymbol{\Phi}^T\boldsymbol{y} + \boldsymbol{a}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{a}$.

O mínimo global pode ser encontrado pelo método dos mínimos quadrados (ou também conhecido como equação normal).

Observe que a função de erro é convexa, com um único mínimo global.

Convexo: curvo ou curvado para fora como o exterior de uma tigela ou esfera ou círculo.

Uma função de valor real definida em um intervalo n-dimensional é denominada ***convexa*** se um segmento de linha entre dois pontos no gráfico da função estiver acima ou no gráfico.

Prova de que formas quadráticas são convexas:

- https://math.stackexchange.com/questions/526657/show-convexity-of-the-quadratic-function
- https://math.stackexchange.com/questions/483339/proof-of-convexity-of-linear-least-squares

A figura da direita mostra um gráfico com linhas de contorno. Uma linha de contorno ou isoline de uma função de duas variáveis é uma curva ao longo da qual a função tem um valor constante. Ou seja, no nosso caso, cada uma das linhas indica curvas que tem o mesmo erro.

Slide 13

Regressão Linear em Python

```
# Import all the necessary libraries.
import numpy as np

# Generate input/ouput (features/labels) values.
N = 100; # Number of observations.

x = 2 * np.random.rand(N, 1)
y = 4 + 3 * x + np.random.randn(N, 1)

# Solve by applying the least-Squares method.
# We use the inv() function from NumPy's Linear Algebra module (np.linalg) to
compute the inverse of a matrix.
# We use dot() method for matrix multiplication.
X_b = np.c_[np.ones((N, 1)), x] # add x0 = 1 to each instance
a_optimum = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)

# Print best solution.
print('a0: %1.4f' % (a_optimum[0][0]))
print('a1: %1.4f' % (a_optimum[1][0]))

a0: 3.8370
a1: 3.0704
```

```
# The equivalent solution using the Scikit-Learn library is given below
# Import the linear regression module form the library.
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression() # instantiate it.
lin_reg.fit(x, y)

print('a0: %1.4f ' % (lin_reg.intercept_[0])) # Value that crosses the y-axis when all
features are equal to 0.
print('a1: %1.4f ' % (lin_reg.coef_[0][0])) # parameters associated with the features.

a0: 3.8370
a1: 3.0704
```

- Perceba que com apenas 100 exemplos, os valores obtidos são bem próximos dos exatos, porém, o ruído torna impossível recuperar os parâmetros exatos da função original.
- Se aumentarmos o número de exemplos conseguimos melhorar a estimação.

numpy.c_ : concatena vetores.

Perceba que com apenas 100 observações (ou exemplos), os valores obtidos são bem próximos dos exatos, porém, o ruído torna impossível recuperar os parâmetros exatos da função original.
Se aumentarmos o número de observações conseguimos melhorar a estimação.

O argumento x passado para o método fit da classe LinearRegression é uma matriz com N x K+1, ou seja, uma matriz com N exemplos como linhas e K+1 atributos como colunas.
Caso sua função hipótese não considere o peso a0, então, durante a instanciação da classe você deve **fit_intercept=False**, pois por default o método fit tenta encontrar o valor de a0.

Slide 14

Desvantagens da forma fechada (Eq. Normal)

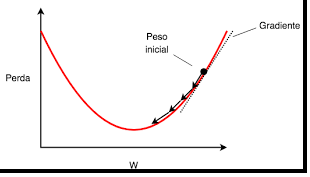
- A solução da ***equação normal*** envolve o cálculo da inversa de $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$, o qual tem complexidade computacional que varia de $O(n^{2.4})$ a $O(n^3)$.
  - **Ex.**: Se o número de ***features*** dobrar, o tempo para cálculo aumenta de $2^{2.4} = 5.3$ a $2^3 = 8$ vezes.
- Dependendo do número de ***exemplos***, N, e ***features***, $\boldsymbol{x}$, a matriz $\boldsymbol{\Phi}$ pode consumir muita memória.
- **Portanto, essa abordagem não é escalonável!**
- Adicionalmente, para irmos além dos modelos lineares (i.e., regressão não-linear) precisamos lidar com o fato de que não existem formas fechadas como a **equação normal**.
- **Solução**: abordagens iterativas
  - Métodos iterativos de busca que façam a atualização dos parâmetros, $\boldsymbol{a}$, à medida que os dados forem sendo apresentados ao modelo.

- A solução da ***equação normal*** envolve o calculo da inversa de $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$, o qual tem complexidade computacional variando entre $O(n^{2.4})$ e $O(n^3)$, dependendo da implementação utilizada.
- Um algoritmo de complexidade O(n^2) tem crescimento quadrático e O(n^3) cúbco.
- $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ é uma matriz $(K+1 \times K+1)$
- Pesquisa em tomografia sísmica envolve um número muito grande de features, da ordem de 10000 features.

Slide 15

Gradiente Descendente

- Algoritmo de otimização genérico capaz de encontrar soluções ótimas para uma ampla gama de problemas.
- Ideia geral é ajustar os parâmetros, $\boldsymbol{a}$, iterativamente, a fim de minimizar a função de erro.
- Utilizado em vários problemas de aprendizado de máquina.
- Escalona melhor do que o método de equação normal para grandes conjuntos de dados.
- Fácil implementação.
- Não é necessário se preocupar com matrizes mal condicionadas (determinante próximo de 0, i.e., quase singulares).



O Gradiente Descendente (GD) é um algoritmo de otimização genérico capaz de encontrar soluções ideais para uma ampla gama de problemas.

A ideia geral do GD é ajustar os parâmetros iterativamente, a fim de minimizar uma função de custo.

Para algoritmos que utilizam o Gradiente Descendente para otimizar os parâmetros do modelo, todas as funções devem ser diferenciáveis.

O Gradiente descendente é utilizado em vários problemas de aprendizado de máquina.

Não precisa se preocupar com matrizes mal-condicionadas, ou seja, com determinante próximo de zero.

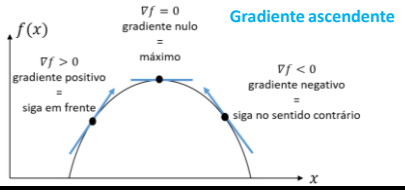
Slide 16

Gradiente Ascendente

- O ***vetor gradiente*** de uma função, $f(x_0, x_1, \dots, x_K)$, em relação aos seus argumentos $x_k, k = 0, \dots, K$, é definido por

$$\nabla f(x_0, x_1, \dots, x_K) = \left[\frac{\partial f(x_0,x_1,\dots,x_K)}{\partial x_0} \quad \frac{\partial f(x_0,x_1,\dots,x_K)}{\partial x_1} \quad \dots \quad \frac{\partial f(x_0,x_1,\dots,x_K)}{\partial x_K}\right]^T,$$

- Onde $\nabla f(x_0, x_1, \dots, x_K)$ é o vetor que aponta a direção em que a função, $f(x_0, x_1, \dots, x_K)$, tem a taxa de aumento/crescimento mais rápida.
- O valor do gradiente em um ponto é um vetor tangente ao ponto.
  - Valor +: máximo à frente. Valor -: máximo atrás. Valor 0: ponto de máximo.
- Portanto, o ***gradiente*** ajuda a encontrar o ***máximo*** da função, $f(x_0, x_1, \dots, x_K)$.
  - Seguindo na direção do gradiente, você chegará ao máximo da função.
- Assim, o algoritmo de otimização iterativo com objetivo de ***maximização*** de $f(x_0, x_1, \dots, x_K)$ é conhecido como ***gradiente ascendente***.



Vocês se lembram das aulas de cálculo, onde vocês aprenderam sobre o gradiente.

∇f => Nabla f

O gradiente pode ser interpretado como a "direção em que uma função tem taxa de aumento mais rápido".

O valor do gradiente em um ponto é um vetor tangente ao ponto. Valores positivos indicam que o aumenta mais rápido esta à frente, já valores negativos indicam que a taxa de aumento mais rápida está para trás. O valor zero indica que estamos sobre o máximo.

Imagine você parado no ponto x de uma função, f, o vetor ∇f diz em qual direção você deve caminhar para aumentar o valor da função f mais rapidamente.

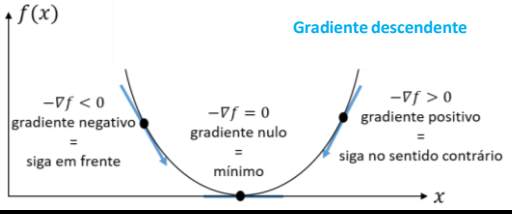
Se você seguir na direção do gradiente, você chegará ao máximo da função.

Note que cada elemento do vetor gradiente aponto para a direção de máxima variação em relação à aquele argumento/parâmetro.

Slide 17

Gradiente Descendente

- Mas e se formos na direção contrária à da taxa de crescimento, dada pelo vetor $\nabla f(x_0, x_1, \dots, x_K)$, ou seja $-\nabla f(x_0, x_1, \dots, x_K)$?
  - Iremos na direção de decrescimento mais rápido da função, $f(x_0, x_1, \dots, x_K)$.
- Portanto, o algoritmo de otimização iterativo com objetivo de ***minimização*** de $f(x_0, x_1, \dots, x_K)$ é conhecido como ***gradiente descendente***.



Vocês se lembram das aulas de cálculo, onde vocês aprenderam sobre o gradiente.

∇f => Nabla f

O gradiente pode ser interpretado como a "direção em que uma função tem taxa de aumento mais rápido".

O valor do gradiente em um ponto é um vetor tangente. Valores positivos indicam que o aumenta mais rápido esta à frente, já valores negativos indicam que a taxa de aumento mais rápida está para trás. O valor zero indica que estamos sobre o máximo.

Imagine você parado no ponto x de uma função, f, o vetor ∇f diz em qual direção você deve caminhar para aumentar o valor da função f mais rapidamente.

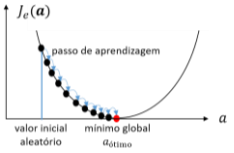
Se você seguir na direção do gradiente, você chegará ao máximo da função.

Slide 18

O Algoritmo do Gradiente do Descendente (GD)

O algoritmo inicializa os parâmetros, $\boldsymbol{a}$, em um ponto aleatório do espaço de parâmetros e então, os atualiza na direção oposta ao do ***gradiente*** até que algum critério de convergência seja atingido, indicando que o mínimo local ou global da função de erro foi encontrado.

$\boldsymbol{a} \leftarrow$ inicializa em um ponto qualquer do espaço de parâmetros
loop até convergir **ou** atingir número máximo de épocas **do**
    for each a_i **in** $\boldsymbol{a}$ **do**
        $a_i \leftarrow a_i - \alpha \frac{\partial J_e(\boldsymbol{a})}{\partial a_i}$



onde α é a ***taxa/passo de aprendizagem*** e $\frac{\partial J_e(\boldsymbol{a})}{\partial a_i}$ é o gradiente da função de erro em relação ao parâmetro a_i. $\nabla J_e(\boldsymbol{a}) = \left[\frac{\partial J_e(\boldsymbol{a})}{\partial a_0} \quad \dots \quad \frac{\partial J_e(\boldsymbol{a})}{\partial a_K}\right]^T$ é o ***vetor do gradiente***.

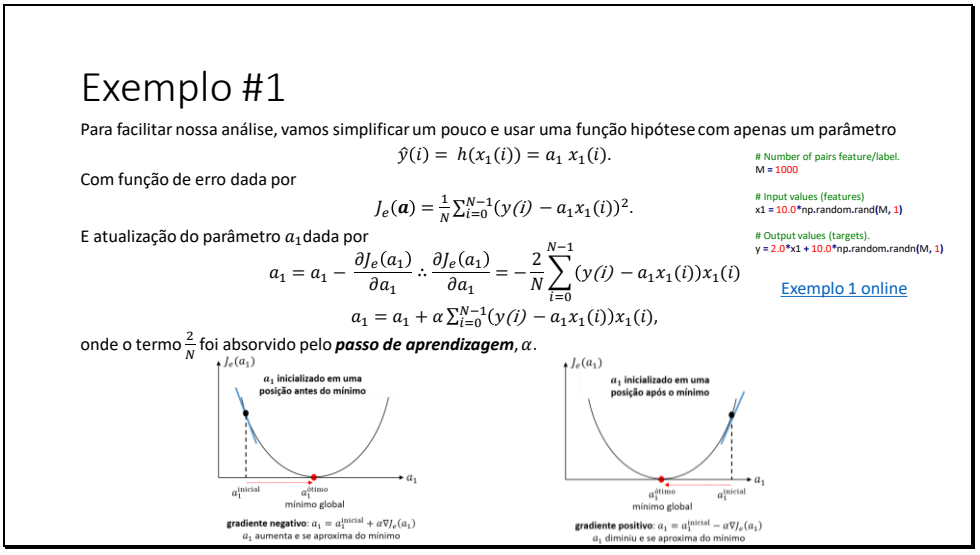
- O ***passo de aprendizagem***, α, pode ser constante ou pode decair com o tempo à medida que o processo de aprendizado prossegue.

OBS.: Os parâmetros, $\boldsymbol{a}$, **devem ser atualizados simultaneamente**, caso contrário o algoritmo apresentará comportamento desconhecido.

Inicializa parâmetros, a, em um ponto aleatório do espaço de parâmetros e então, atualiza os parâmetros na direção oposta ao do gradiente até que algum critério de convergência seja atingido, indicando que o mínimo global da função de erro/custo foi encontrado.

Taxa de aprendizado: tamanho dos passos/deslocamento dado na direção oposta ao gradiente.

Slide 19



Exemplo: linear_regression_with_gradient_descente_exemplo1.ipynb

Figuras: linear_regression_with_gradient_descente_exemplo1.m

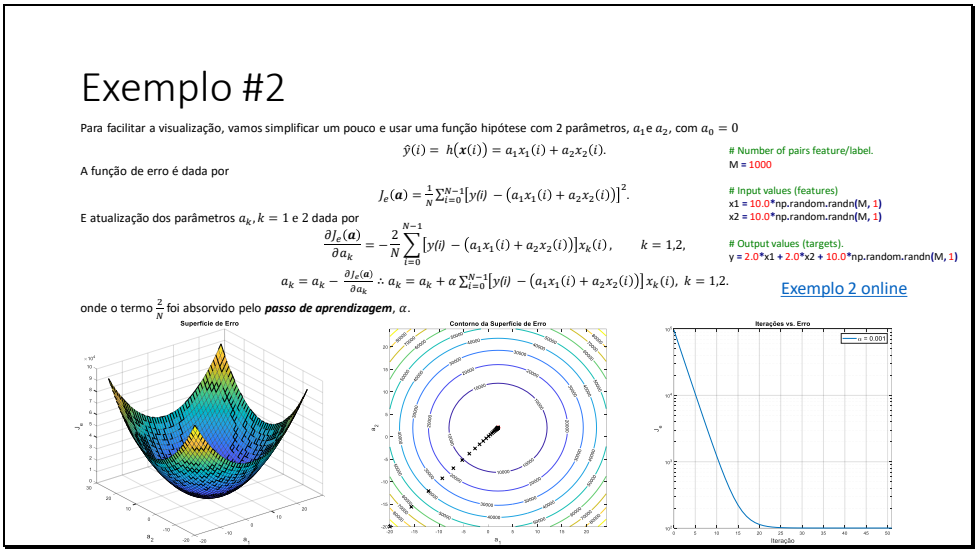
Essa simplificação tem como objetivo facilitar a visualização dos resultados.

Na figura da esquerda, veja que a tangente no ponto $a_1^{inicial}$, tem inclinação negativa (ou seja, o gradiente é negativo), indicando que o valor de $a_1^{inicial}$ deve ser aumentado, indo para a direita e se aproximando de $a_1^{ótimo}$

Na figura da direita, veja que a tangente no ponto $a_1^{inicial}$, tem inclinação positiva (ou seja, o gradiente é positivo), indicando que o valor de $a_1^{inicial}$ deve ser diminuído, indo para a esquerda e se aproximando de $a_1^{ótimo}$

O passo de aprendizado, α, pode ser um valor constante ou pode decair com o tempo à medida que o processo de aprendizado prossegue.

Slide 20



Exemplo: linear_regression_with_gradient_descente_exemplo2.ipynb

Online: https://colab.research.google.com/github/zz4fap/tp555-machine-learning/blob/master/exemplos/linear_regression_with_gradient_descent_exemplo2.ipynb

Figuras: linear_regression_with_gradient_descente_exemplo2.m

Nesse exemplo, o vetor 𝒂 inicial, é inicializado com os valores [-20; -20] e vemos que o algoritmo caminha progressivamente em direção ao mínimo global mostrado pelo asterisco em vermelho (esse ponto foi calculado com o método da equação normal). O passo de aprendizado é feito igual a 0.001 e como podemos ver o algoritmo converge à partir da iteração número 25.

A figura do meio mostra a trajetória realizada pelo algoritmo até a convergência.

Vejam que o algoritmo converge lentamente e portanto, é possível aumentar o passo de aprendizagem.

O passo de aprendizado, α, pode ser um valor constante ou pode decair com o tempo à medida que o processo de aprendizado prossegue.

Slide 21

Exemplo #3

Função hipótese com 2 parâmetros, a_0e a_1,

$$\hat{y}(i) = h(\boldsymbol{x}(i)) = a_0 + a_1 x_1(i).$$

A função de erro é dada por

$$J_e(\boldsymbol{a}) = \frac{1}{N}\sum_{i=0}^{N-1}\left[y(i) - \left(a_0 + a_1 x_1(i)\right)\right]^2.$$

E atualização dos parâmetros $a_k, k = 0$ e 1 dada por

$$\frac{\partial J_e(\boldsymbol{a})}{\partial a_k} = -\frac{2}{N}\sum_{i=0}^{N-1}\left[y(i) - \left(a_0 + a_1 x_1(i)\right)\right]x_k(i), \qquad k = 0, 1,$$

$$a_k = a_k - \frac{\partial J_e(\boldsymbol{a})}{\partial a_k} \therefore a_k = a_k + \alpha\sum_{i=0}^{N-1}\left[y(i) - \left(a_0 + a_1 x_1(i)\right)\right]x_k(i),\ k = 0,1,$$

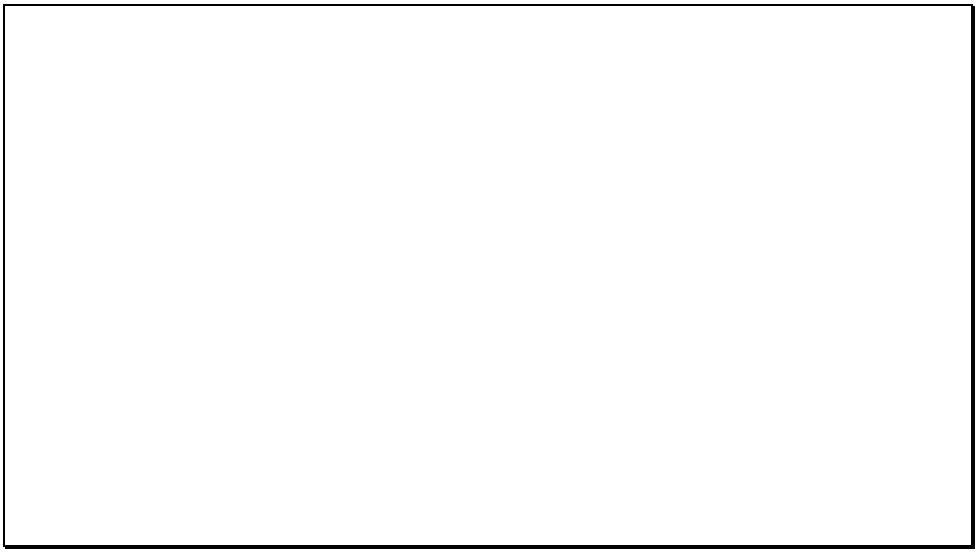
onde $x_0(i) = 1\ \forall i$.

Exemplo 3 online

https://colab.research.google.com/github/zz4fap/tp555-machine-learning/blob/master/exemplos/linear_regression_with_gradient_descent_exemplo3.ipynb

Para executar esse exemplo é necessário instalar a biblioteca ffmpeg com o comando conda install ffmpeg

Slide 22



Slide 23

Avisos

- Aulas (vídeos) mais curtas com listas sobre os assuntos discutidos.
- Alguns estudos dirigidos também com listas.

Slide 24

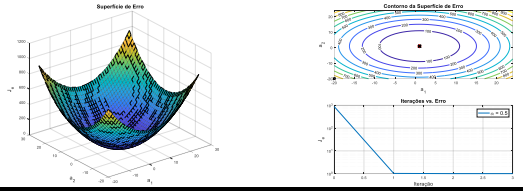
Recapitulando

- Aprendemos o que é regressão linear e como realizá-la
  - de forma fechada através da equação normal,
  - e iterativamente através do gradiente descendente.
- Analisamos 3 exemplos feitos em Python e executados no Jupyter e verificamos que a escolha do passo de aprendiozagem é muito impiortante para a convergência do gradiente descendente.
- Na aula de hoje nós falamos sobre
  - a escolha do passo de aprendizagem,
  - diferentes versões do gradiente descendente,
  - e sobre pré-processamneto do conjunto de treinamento.

Slide 25

Espaço de Busca dos Parâmetros

- Treinar um modelo significa procurar por uma combinação de pesos/parâmetros, $\boldsymbol{\alpha}$, do modelo que minimizem uma ***função de erro***.
- É uma procura no ***espaço de parâmetros do modelo***: é o espaço formado por todas as possíveis combinações de pesos.
- Quanto mais parâmetros um modelo tem, mais dimensões o espaço de parâmetros terá e mais difícil será a pesquisa.
  - **Exemplo**: Encontrar os pesos de 300 atributos significa um espaço com 300 dimensões.
- No caso da regressão linear, a procura é mais simples pois o ponto ótimo está sempre no fundo do "vale" ou da "tigela".

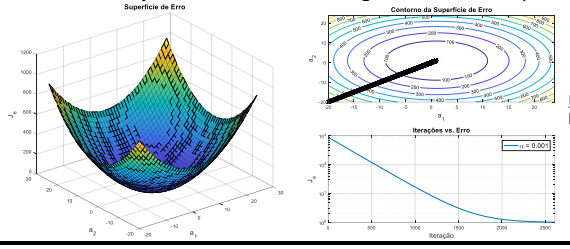


Treinar um modelo significa procurar uma combinação de parâmetros do modelo que minimizem uma função de custo (no conjunto de treinamento). É uma pesquisa no espaço de parâmetros do modelo: quanto mais parâmetros um modelo tem, mais dimensões esse espaço tem e mais difícil é a pesquisa: procurar uma agulha em um palheiro de 300 dimensões é muito mais difícil do que em três dimensões. Felizmente, como a função de custo é convexa no caso de regressão linear, a agulha está simplesmente no fundo da tigela.

Slide 26

Escolha do Passo de Aprendizagem

- Enquanto a direção para o mínimo é determinada pelo vetor gradiente da função de erro, o ***passo de aprendizagem*** determina o quão grande o passo é dado naquela direção.
- Portanto, a **escolha do passo de aprendizagem é muito importante**:
  - Caso seja muito pequeno, a convergência do algoritmo levará muito tempo.
  - **Exemplo**: com $\alpha = 0.001$ atinge o valor ótimo após mais de 2500 iterações/épocas.



Exemplo 4 online
linear_regression_selecting_the_learning_rate.ipynb

Se o passo de aprendizagem for muito pequeno, o algoritmo precisará passar por muitas iterações para convergir, o que levará muito tempo.

Por outro lado, se ele for muito grande, você pode pular o vale e acabar do outro lado, possivelmente até mais alto do que antes. Isso pode fazer o algoritmo divergir, com valores cada vez maiores, falhando em encontrar uma boa solução.

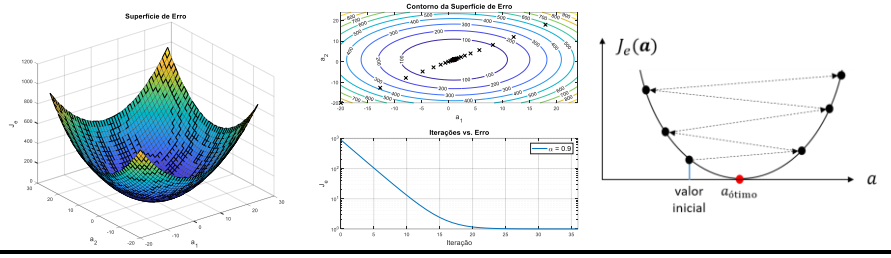
Assim, o passo de aprendizagem deve ser experimentado/explorado para encontrar o melhor valor que acelere a descida do gradiente.

Exemplo: linear_regression_selecting_the_learning_rate.ipynb
Link: https://colab.research.google.com/github/zz4fap/tp555-machine-learning/blob/master/exemplos/regression/linear_regression_selecting_the_learning_rate.ipynb

Slide 27

Escolha do Passo de Aprendizagem

- Caso o ***passo de aprendizagem*** seja muito grande, o algoritmo pode nunca convergir.
- O algoritmo fica "pulando" de um lado para o outro do vale até que converge, por sorte.
- Em alguns casos, a cada iteração o algoritmo "pula" para um valor mais alto que antes, divergindo.



Enquanto a direção em direção ao mínimo é determinada a partir do gradiente da função de erro, a taxa de aprendizado determina o quão grande um passo é dado nessa direção.

Se o passo de aprendizagem for muito pequeno, o algoritmo precisará passar por muitas iterações para convergir, o que levará muito tempo.

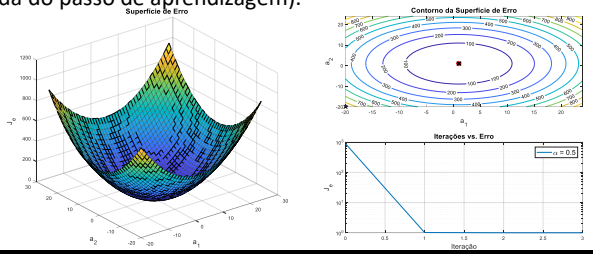
Por outro lado, se ele for muito grande, você pode pular o vale e acabar do outro lado, possivelmente até mais alto do que antes. Isso pode fazer o algoritmo divergir, com valores cada vez maiores, falhando em encontrar uma boa solução.

Assim, o passo de aprendizagem deve ser experimentado/explorado para encontrar o melhor valor que acelere a descida do gradiente.

Slide 28

Escolha do Passo de Aprendizagem

- O valor ***passo de aprendizagem*** deve ser ***experimentado/explorado*** para se encontrar o ***valor ótimo*** que acelere a descida do gradiente de forma ***estável*** (ou seja, acelere a convergência).
- Exemplo abaixo converge para o mínimo local em apenas 3 iterações.
- **OBS**.: Passos largos durante as iterações iniciais e curtos conforme o algoritmo se aproxima do mínimo podem acelerar a convergência (esquemas de variação programada do passo de aprendizagem).



Se o passo de aprendizagem for muito pequeno, o algoritmo precisará passar por muitas iterações para convergir, o que levará muito tempo.

Por outro lado, se ele for muito grande, você pode pular o vale e acabar do outro lado, possivelmente até mais alto do que antes. Isso pode fazer o algoritmo divergir, com valores cada vez maiores, falhando em encontrar uma boa solução.

Assim, o passo de aprendizagem deve ser experimentado/explorado para encontrar o melhor valor que acelere a descida do gradiente.

Slide 29

Como depurar o algoritmo do GD?

Uma das maneiras de se ***depurar*** (principalemnte quando não é possível se plotar o gráfico de contorno) o algoritmo do ***gradiente descendente*** é plotar o gráfico da função de custo em função do número de épocas/iterações.

- Figura A ⇒ Passo ótimo: converge rapidamente
  - Erro diminui rapidamente nas primeiras épocas e depois diminui quase que a uma taxa constante.
  - Convergência pode ser declarada quando o erro entre duas épocas subsequentes for menor do que um limiar pré-definido (e.g., 1e-3).
- Figura B ⇒ Passo pequeno demais: convergência lenta.
- Figuras C e D ⇒ Passo grande demais: divergência.

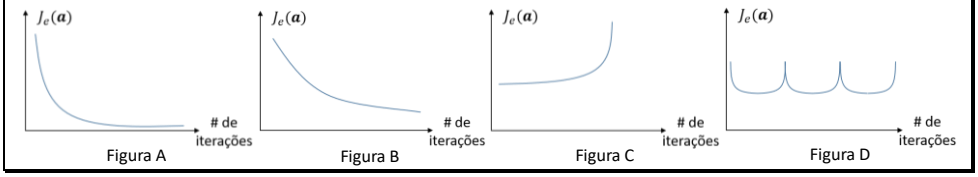


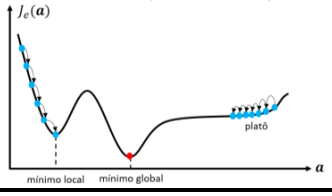
Figura A Figura B Figura C Figura D

Como você consegue debugar/depurar o algoritmo do gradiente descendente quando não é possível se plotar o gráfico de contorno e verificar o caminho seguido pelo algoritmo?

Slide 30

Escolha do Passo de Aprendizagem

- A superfície de funções de erro utilizando o ***erro quadrático médio*** para modelos de regressão linear, são sempre ***funções convexas***, se assemelhando a um vale ou a uma tigela.
  - **Implicações**: Não existem mínimos locais, apenas um mínimo global. É também uma função contínua com uma inclinação que nunca muda abruptamente.
  - **Consequência**: o gradiente descendente garantidamente se aproxima do mínimo global (dado que você espere tempo suficiente e que o ***passo de aprendizagem*** não seja grande demais).
- Porém, nem todas as superfícies de erro se parecem com vales ou tigelas, ou seja, são convexas. Algumas podem ter vários mínimos locais, platôs, e todo tipo de "terreno irregular", dificultando a convergência para o mínimo global.
- Além disso, como dito antes, passos grandes podem desestabilizar o algoritmo e passos muito pequenos aumentar demais o tempo de convergência.

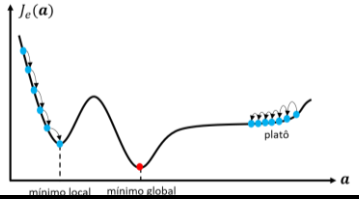


Platô: terreno elevado e plano.

Slide 31

Desafios do Gradiente Descente

- A figura mostra dois dos principais desafios encontrados pelo algortimo do Gradiente Descente:
  - Se a inicialização aleatória dos pesos iniciar o algoritmo à esquerda, ele convergirá para um mínimo local, que não é tão bom quanto o mínimo global.
  - Se começar à direita, levará muito tempo para atravessar o platô (gradiente próximo de zero pois a inclinação é próxima de 0 graus) e, se ele parar muito cedo, nunca alcançará o mínimo global.
- Dado que o passo de aprendizagem seja grande o suficiente, como garantir que o mínimo encontrado é o global e não um mínimo local?
  - O que se faz é treinar o modelo várias vezes, sempre inicializando os pesos aleatoriamente, com a esperança de que em alguma dessas vezes ele inicialize mais próximo do mínimo global.



Platô: terreno elevado e plano.

A Figura acima mostra os dois principais desafios do Gradiente Descente: se a inicialização aleatória iniciar o algoritmo à esquerda, convergirá para um mínimo local, que não é tão bom quanto o mínimo global. Se começar à direita, levará muito tempo para atravessar o platô (gradiente próximo de zero pois a inclinação é próxima de 0 graus) e, se você parar muito cedo, nunca alcançará o mínimo global.

Outro tipo de terreno irregular é o terreno em forma de sela (de cavalo). Um ponto de sela é o ponto sobre uma superfície no qual a declividade é nula, mas não se trata de um extremo local (máximo ou mínimo). É o ponto sobre uma superfície na qual a elevação é máxima numa direção e mínima na outra direção (por exemplo, na direção perpendicular).

Dado que o passo de aprendizagem seja grande o suficiente como garantir que o mínimo encontrado é o global e não um mínimo local?
Muitas vezes o que se faz é treinar o modelo várias vezes, sempre inicializando os pesos de pontos diferentes, ou seja, aleatoriamente.

Slide 32

Escolha do Passo de Aprendizagem

- Como vimos, além da inicialização dos pesos do modelo a escolha do ***passo de aprendizagem*** é muito importante para a convergência do GD.
- Passos muito grandes fazem com que o algoritmo aprenda rápido demais ao custo de um modelo final que seja sub-ótimo ou que o treinamento se torne instável (oscilação).
- Passos muito pequenos resultam num longo treinamento, podendo o algoritmo, por exemplo, ficar preso em um mínimo local ou mesmo nunca atingir um mínimo.
- Portanto, como podemos ajustar o ***passo de aprendizagem***?

Um valor muito pequeno pode resultar em um longo processo de treinamento que pode ficar preso.
Um valor muito alto pode resultar na aprendizagem de um conjunto subótimo de pesos rápido demais ou em um processo de treinamento instável.

A maneira pela qual a taxa de aprendizado muda com o tempo (iteração/época) é chamada de cronograma/programa da taxa de aprendizado ou decaimento da taxa de aprendizado.

Alguns tipos de esquema para ajuste do passo de aprendizagem são:

- **Decaimento por etapas ou degraus**: reduz a taxa de aprendizado de algum fator a cada número pré-definido de iterações ou épocas. Os valores típicos são utilizados para reduzir a taxa de aprendizado pela metade a cada número pré-definido de épocas. Esses números dependem muito do tipo de problema e do modelo. Uma heurística que você pode ver na prática é observar o erro de validação durante o treinamento com uma taxa de aprendizado fixa e reduzir a taxa de aprendizado em uma constante (por exemplo, 0,5) sempre que o erro de validação parar de decrescer.
- **Decaimento exponencial**: tem a forma matemática $\alpha = \alpha 0\, e^{(-kt)}$, onde $\alpha 0$, k são hiperparâmetros e t é o número da iteração (mas você também pode usar o número de épocas).
- **Decaimento temporal**: tem a forma matemática $\alpha = \alpha 0 / (1 + kt)$, onde a0, k são hiperparâmetros e t é o número da iteração.

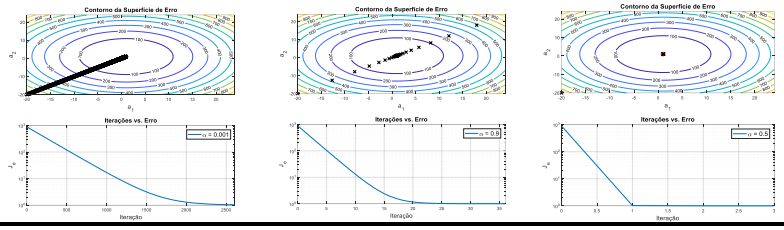
Modificação Adaptativa:
As abordagens anteriores manipulam a taxa de aprendizado global e igualmente para todos os parâmetros. Ajustar a taxa de aprendizado é um processo caro, muito trabalho foi desenvolvido para a criação de métodos que possam ajustar adaptativamente as taxas de aprendizado, e até fazê-lo por parâmetro. Muitos desses métodos ainda podem exigir outras

configurações de hiperparâmetro, mas o argumento é que eles são bem-comportados para uma faixa mais ampla de valores de hiperparâmetro do que o ajuste do passo de aprendizado.

Slide 33

Ajuste do passo de aprendizagem

- Uma maneira de se ajustar o passo é através do **ajuste manual**, onde o ajuste do passo de aprendizado é feito através de tentativa e erro até que um passo ótimo seja encontrado.
  - A desvantagem é que isso pode levar muito tempo, imagine um conjunto de treinamento com milhões de exemplos e dezenas ou até mesmo centenas de pesos a serem ajustados e você ter que esperar até que o treinamento termine.
- Existem outras formas de se ajustar o passo mais eficientemente?



A maneira pela qual a taxa de aprendizado muda com o tempo (iteração/época) é chamada de cronograma/programa da taxa de aprendizado ou decaimento da taxa de aprendizado.

Alguns tipos de esquema para ajuste do passo de aprendizagem são:

- **Decaimento por etapas ou degraus**: reduz a taxa de aprendizado de algum fator a cada número pré-definido de iterações ou épocas. Os valores típicos são utilizados para reduzir a taxa de aprendizado pela metade a cada número pré-definido de épocas. Esses números dependem muito do tipo de problema e do modelo. Uma heurística que você pode ver na prática é observar o erro de validação durante o treinamento com uma taxa de aprendizado fixa e reduzir a taxa de aprendizado em uma constante (por exemplo, 0,5) sempre que o erro de validação parar de decrescer.
- **Decaimento exponencial**: tem a forma matemática $\alpha = \alpha 0\, e^{(-kt)}$, onde $\alpha 0$, k são hiperparâmetros e t é o número da iteração (mas você também pode usar o número de épocas).
- **Decaimento temporal**: tem a forma matemática $\alpha = \alpha 0 / (1 + kt)$, onde a0, k são hiperparâmetros e t é o número da iteração.

Modificação Adaptativa:

As abordagens anteriores manipulam a taxa de aprendizado global e igualmente para todos os parâmetros. Ajustar a taxa de aprendizado é um processo caro, muito trabalho foi desenvolvido para a criação de métodos que possam ajustar adaptativamente as taxas de aprendizado, e até fazê-lo por parâmetro. Muitos desses métodos ainda podem exigir outras configurações de hiperparâmetro, mas o argumento é que eles são bem-comportados para uma faixa mais ampla de valores de hiperparâmetro do que o ajuste do passo de aprendizado.

Slide 34

Ajuste do passo de aprendizagem

- Seria interessante que no incio do treinamento o algoritmo desse passos largos e aprendesse mais rápido e conforme ele se aproximasse do mínimo, gostaríamos que ele desse passos mais curtos para não ultrapassá-lo.
- Portanto, para se obter uma convergência mais rápida, evitar oscilações e ficar preso em mínimos locais, o passo de aprendizagem é geralmente variado durante o treinamento, de acordo com um ***esquema de variação (em etapas) do passo de aprendizagem***.

A maneira pela qual a taxa de aprendizado muda com o tempo (iteração/época) é chamada de cronograma/programa da taxa de aprendizado ou decaimento da taxa de aprendizado.

Alguns tipos de esquema para ajuste do passo de aprendizagem são:

- **Decaimento por etapas ou degraus**: reduz a taxa de aprendizado de algum fator a cada número pré-definido de iterações ou épocas. Os valores típicos são utilizados para reduzir a taxa de aprendizado pela metade a cada número pré-definido de épocas. Esses números dependem muito do tipo de problema e do modelo. Uma heurística que você pode ver na prática é observar o erro de validação durante o treinamento com uma taxa de aprendizado fixa e reduzir a taxa de aprendizado em uma constante (por exemplo, 0,5) sempre que o erro de validação parar de decrescer.
- **Decaimento exponencial**: tem a forma matemática $\alpha = \alpha 0\, e^{(-kt)}$, onde $\alpha 0$, k são hiperparâmetros e t é o número da iteração (mas você também pode usar o número de épocas).
- **Decaimento temporal**: tem a forma matemática $\alpha = \alpha 0 / (1 + kt)$, onde a0, k são hiperparâmetros e t é o número da iteração.

Modificação Adaptativa:

As abordagens anteriores manipulam a taxa de aprendizado global e igualmente para todos os parâmetros. Ajustar a taxa de aprendizado é um processo caro, muito trabalho foi desenvolvido para a criação de métodos que possam ajustar adaptativamente as taxas de aprendizado, e até fazê-lo por parâmetro. Muitos desses métodos ainda podem exigir outras configurações de hiperparâmetro, mas o argumento é que eles são bem-comportados para uma faixa mais ampla de valores de hiperparâmetro do que o ajuste do passo de aprendizado.

Slide 35

Esquemas de variação de α

Os esquemas para variação podem ser divididos em 2 categorias: variação programada e adaptativa.

- **Variação ou decaimento programado**: o passo de aprendizagem tem seu valor diminuído ao longo do tempo (época), ou seja, ao longo do processo de treinamento.
  - Uma desvantagem é que os parâmetros desses esquemas devem ser manualmente ajustados previamente e dependem do problema e do modelo adotado. Portanto, cai-se de certa forma, no problema da tentativa e erro.
  - O ajuste dos parâmetros dos esquemas de variação programada é custoso.
  - Outra desvantagem é que o mesmo passo de aprendizado é aplicado ao ajuste de todos os pesos.
  - **Exemplos**: Momentum, Decaimento exponencial, Decaimento por etapas, etc.

A maneira pela qual a taxa de aprendizado muda com o tempo (iteração/época) é chamada de cronograma/programa da taxa de aprendizado ou decaimento da taxa de aprendizado.

Alguns tipos de esquema para ajuste do passo de aprendizagem são:

- **Decaimento por etapas ou degraus**: reduz a taxa de aprendizado de algum fator a cada número pré-definido de iterações ou épocas. Os valores típicos são utilizados para reduzir a taxa de aprendizado pela metade a cada número pré-definido de épocas. Esses números dependem muito do tipo de problema e do modelo. Uma heurística que você pode ver na prática é observar o erro de validação durante o treinamento com uma taxa de aprendizado fixa e reduzir a taxa de aprendizado em uma constante (por exemplo, 0,5) sempre que o erro de validação parar de decrescer.
- **Decaimento exponencial**: tem a forma matemática $\alpha = \alpha 0\, e^{(-kt)}$, onde $\alpha 0$, k são hiperparâmetros e t é o número da iteração (mas você também pode usar o número de épocas).
- **Decaimento temporal**: tem a forma matemática $\alpha = \alpha 0 / (1 + kt)$, onde a0, k são hiperparâmetros e t é o número da iteração.

Modificação Adaptativa:

As abordagens anteriores manipulam a taxa de aprendizado global e igualmente para todos os parâmetros. Ajustar a taxa de aprendizado é um processo caro, muito trabalho foi desenvolvido para a criação de métodos que possam ajustar adaptativamente as taxas de aprendizado, e até fazê-lo por parâmetro. Muitos desses métodos ainda podem exigir outras configurações de hiperparâmetro, mas o argumento é que eles são bem-comportados para uma faixa mais ampla de valores de hiperparâmetro do que o ajuste do passo de aprendizado.

Slide 36

Esquemas de variação de α

- **Variação ou decaimento adaptativo**: passo é adaptativamente ajustado de acordo com a performance do modelo além disso possui passos diferentes para cada parâmetro do modelo e os atualiza independentemente.
  - Os passos são atualizados de acordo com valores obtidos pelo modelo (e.g., média movente dos gradientes).
  - Na maioria dos casos, não é necessário se ajustar manualmente nenhum parâmetro como no caso dos esquemas de variação programada.
  - E quando existe algum parâmetro a ser ajustado o esquema normalmente funciona muito bem para uma grande gama de valores.
  - **Exemplos**: Adam, Adagrad, RMSprop, etc.

A maneira pela qual a taxa de aprendizado muda com o tempo (iteração/época) é chamada de cronograma/programa da taxa de aprendizado ou decaimento da taxa de aprendizado.

Alguns tipos de esquema para ajuste do passo de aprendizagem são:

- **Decaimento por etapas ou degraus**: reduz a taxa de aprendizado de algum fator a cada número pré-definido de iterações ou épocas. Os valores típicos são utilizados para reduzir a taxa de aprendizado pela metade a cada número pré-definido de épocas. Esses números dependem muito do tipo de problema e do modelo. Uma heurística que você pode ver na prática é observar o erro de validação durante o treinamento com uma taxa de aprendizado fixa e reduzir a taxa de aprendizado em uma constante (por exemplo, 0,5) sempre que o erro de validação parar de decrescer.
- **Decaimento exponencial**: tem a forma matemática $\alpha = \alpha 0\, e^{(-kt)}$, onde $\alpha 0$, k são hiperparâmetros e t é o número da iteração (mas você também pode usar o número de épocas).
- **Decaimento temporal**: tem a forma matemática $\alpha = \alpha 0 / (1 + kt)$, onde a0, k são hiperparâmetros e t é o número da iteração.

Modificação Adaptativa:

As abordagens anteriores manipulam a taxa de aprendizado global e igualmente para todos os parâmetros. Ajustar a taxa de aprendizado é um processo caro, muito trabalho foi desenvolvido para a criação de métodos que possam ajustar adaptativamente as taxas de aprendizado, e até fazê-lo por parâmetro. Muitos desses métodos ainda podem exigir outras configurações de hiperparâmetro, mas o argumento é que eles são bem-comportados para uma faixa mais ampla de valores de hiperparâmetro do que o ajuste do passo de aprendizado.

Slide 37

Versões do Gradiente Descendente

Existem 3 diferentes versões para a implementação do algoritmo do Gradiente Descendente: Batelada, Estocástico e Mini-Batch.

- **Batelada (do inglês *batch*)**: a cada iteração do algoritmo, ***todos*** os exemplos de treinamento são considerados no processo de treinamento do modelo. Esta versão foi utilizada nos exemplos 1, 2 e 3.

$$a_k = a_k + \alpha \sum_{i=0}^{N-1} \left[y(i) - \left(a_1 x_1(i) + a_2 x_2(i) \right) \right] x_k(i), \; k = 1, \ldots, K$$

  - Utilizado quando se possui previamente todos os atributos e rótulos de treinamento, ou seja, o conjunto de treinamento.
  - **Convergência garantida**, dado que o passo de aprendizagem seja pequeno o suficiente.
  - **Convergência pode ser bem lenta**, dado que o modelo é apresentado a todos os exemplos a cada época.

Batch: usa todos os exemplos de treinamento a cada iteração. Como resultado, é muito lento em conjuntos de treinamento muito grandes.

A versão **online** seleciona apenas uma instância aleatória no conjunto de treinamento a cada etapa e calcula os gradientes com base apenas nessa única instância. Obviamente, isso torna o algoritmo muito mais rápido, pois possui muito poucos dados para manipular a cada iteração. Por outro lado, devido à sua natureza estocástica (ou seja, aleatória), esse algoritmo é muito menos regular do que a descida do gradiente em lote: em vez de diminuir suavemente até atingir o mínimo, a função de custo irá saltar para cima e para baixo, diminuindo apenas em média .

Mini-batch: em cada iteração, em vez de calcular os gradientes com base no conjunto de treinamento completo (como no Batch) ou com base em apenas uma instância (como no GD estocástico), o mini-batch GD calcula os gradientes em pequenos conjuntos aleatórios de instâncias chamados mini-batches.

Época
Uma época é quando todo o conjunto de dados (exemplos) de treinamento é utilizado no treinamento do modelo.

Iterações
Iteração corresponde a um batch apresentado ao modelo.
Conta o número de batches necessários para concluir uma época, caso cada batch seja menor do que o conjunto de treinamento.

Tamanho do batch

Número total de exemplos de treinamento presentes em um único batch que será utilizado durante uma iteração de treinamento.
Se um batch conter todos os exemplos de treinamento então cada iteração é igual a uma época.

Slide 38

Tipos de Gradiente Descendente

- **Gradiente Descendente Estocástico**: também conhecido como ***online*** ou ***incremental*** (exemplo-a-exemplo). Os pesos do modelo são atualizados a cada novo exemplo de treinamento.

$$a_k = a_k + \alpha\left[y(i) - \left(a_1 x_1(i) + a_2 x_2(i)\right)\right] x_k(i),\ k = 1, \ldots, K$$

  - Aproxima o gradiente através de uma ***estimativa estocástica***: aproximação através do gradiente calculado com um único exemplo de treinamento.
  - Pode ser utilizado quando os atributos e rótulos são obtidos sequencialmente, ou seja, de forma online, exemplo a exemplo.
  - Ou quando o conjunto de treinamento é muito grande. Nesse caso, escolhe-se aleatoriamente um par atributo/rótulo a cada iteração (i.e., atualização dos pesos).
  - **Convergência mais rápida**.
  - **Porém, ela não é garantida** com um passo de aprendizagem fixo. O algoritmo pode oscilar em torno do mínimo sem nunca convergir para o valores ótimos.
  - O uso de esquemas para variação do passo de aprendizagem garantem a convergência.

Métodos de ***aproximação estocástica*** são uma família de métodos iterativos normalmente usados para problemas de procura de raízes ou para problemas de otimização.

Pode ser considerada como uma aproximação estocástica da otimização do gradiente descendente, uma vez que substitui o valor do gradiente real (calculado a partir de todo o conjunto de dados) por uma estimativa do mesmo (calculado a partir de um subconjunto de exemplos selecionado aleatoriamente).

Slide 39

Tipos de Gradiente Descendente

- **Mini-batch**: é um meio-termo entre as duas versões anteriores. O conjunto de treinamento é dividido em vários subconjuntos (mini-batches) com elementos aleatórios (i.e., par atributo/rótulo), onde os pesos do modelo são ajustados a cada mini-batch.

$$a_k = a_k + \alpha \sum_{i=0}^{MB-1} \left[y(i) - \left(a_1 x_1(i) + a_2 x_2(i) \right) \right] x_k(i), \; k = 1, \ldots, K$$

onde MB é o tamanho do mini-batch.

- Pode ser visto como uma generalização das 2 versões anteriores:
  - Caso $MB =$ N, então se torna o GD em batelada.
  - Caso $MB = 1$, então se torna o GD estocástico.
- Tem convergência mais rápida do que o GD em batelada mas mais lenta do que o GD estocástico.
- Convergência depende do tamanho do mini-batch.
- Pode usar esquemas de variação do passo de aprendizagem para melhorar a convergência.

Batch: usa todos os exemplos de treinamento a cada iteração. Como resultado, é muito lento em conjuntos de treinamento muito grandes.

A versão **online** seleciona apenas uma instância aleatória no conjunto de treinamento a cada etapa e calcula os gradientes com base apenas nessa única instância. Obviamente, isso torna o algoritmo muito mais rápido, pois possui muito poucos dados para manipular a cada iteração. Por outro lado, devido à sua natureza estocástica (ou seja, aleatória), esse algoritmo é muito menos regular do que a descida do gradiente em lote: em vez de diminuir suavemente até atingir o mínimo, a função de custo irá saltar para cima e para baixo, diminuindo apenas em média .

Mini-batch: em cada iteração, em vez de calcular os gradientes com base no conjunto de treinamento completo (como no Batch) ou com base em apenas uma instância (como no GD estocástico), o mini-batch GD calcula os gradientes em pequenos conjuntos aleatórios de instâncias chamados mini-batches.

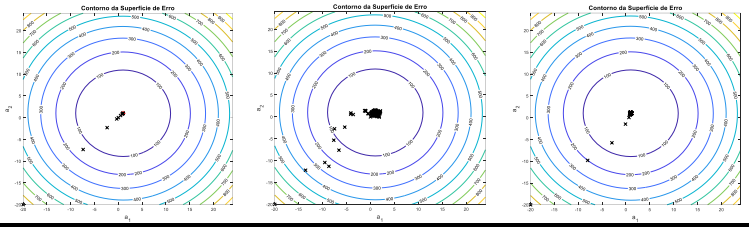
Slide 40

Comparação dos tipos de GD

- Todos se aproximam do mínimo, mas o ***batch*** caminha diretamente em linha reta para lá.
- Enquanto ***SGD*** e o ***mini-batch*** continuam a caminhar ao redor do mínimo.
- O progresso do ***mini-batch*** é menos irregular do que com o ***SGD***, mas depende do tamanho do mini-batch.
- Mas não se esqueça, o ***batch*** leva muito tempo para executar cada época enquanto ***SGD*** e **mini-batch** também alcançariam o mínimo caso uma boa estratégia para ajuste do passo de aprendizagem fosse usada.

Função objetivo
x1 = randn(M, 1)
x2 = randn(M, 1)
y = x1 + x2 + randn(M,1)

Batch ***Gradiente Descendente Estocástico*** ***Mini-Batch (MB size: 100)***



O batch caminha diretamente em linha reta para o mínimo.

O progresso do mini-batch no espaço de parâmetros é menos irregular do que com o SGD, especialmente com mini-batches muito grandes.

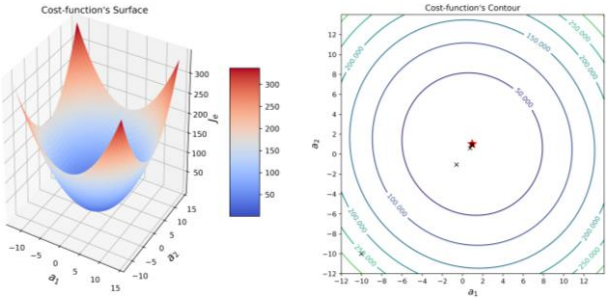
No entanto, não se esqueça que o batch leva muito tempo para executar cada época, e o SGD e o mini-batch também alcançariam o mínimo se você usasse uma boa estratégia para modificação do passo de aprendizagem.

```
x1 = randn(M, 1);
x2 = randn(M, 1);
y = x1 + x2 + randn(M,1);
```

Slide 41

Implementação: Gradiente Descendente por Batelada

```
import numpy as np

# Define the number of examples.
M = 1000

# Generate target function.
x1 = np.random.randn(M, 1)
x2 = np.random.randn(M, 1)
y = x1 + x2 + np.random.randn(M, 1)

# Concatenate both column vectors, x1 and x2.
X = np.c_[x1, x2]

# Constant learning rate.
eta = 0.1
# Number of iterations.
n_iterations = 1000

# Random initialization.
a = np.random.randn(2,1)

# Batch gradient-descent loop.
for iteration in range(n_iterations):
    gradients = -2/M * X.T.dot(y - X.dot(a))
    a = a - eta * gradients
```



- Segue diretamente para o mínimo global.
- Atinge o mínimo em 4 épocas.
- Nesse caso específico, segue linha reta entre a_0 e a_1 pois a taxa de decrescimento da superfície de erro é igual para os dois parâmetros (contornos são circulares).
- Não fica "*oscilando*" em torno do mínimo após alcançá-lo.
- Algoritmo para no mínimo pois o vetor gradiente no ponto ótimo é praticamente nulo.

Exemplo Batch GD

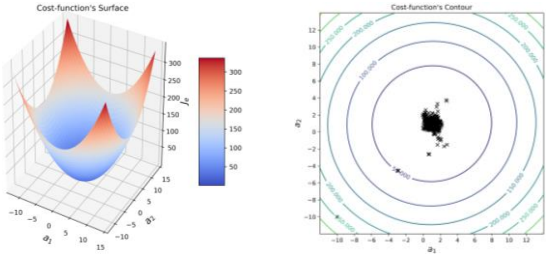
Não fica "*oscilando*" ou "*ricocheteando*" em torno do mínimo após chegar próximo dele.

Exemplo: batch_gradient_descent_with_figures.ipynb

Slide 42

Implementação: Gradiente Descendente Estocástico

```
import numpy as np

# Define the number of examples.
M = 1000

# Generate target function.
x1 = np.random.randn(M, 1)
x2 = np.random.randn(M, 1)
y = x1 + x2 + np.random.randn(M, 1)

# Concatenate both column vectors, x1 and x2.
X = np.c_[x1, x2]

# Number of epochs.
n_epochs = 1
# Constant learning rate.
alpha = 0.1

# Random initialization of parameters.
a = np.random.randn(2,1)
# Stocastic gradient-descent loop.
for epoch in range(n_epochs):
    for i in range(M):
        random_index = np.random.randint(M)
        xi = X[random_index:random_index+1]
        yi = y[random_index:random_index+1]
        gradients = -2*xi.T.dot(yi - xi.dot(a))
        a = a - alpha * gradients
```



Exemplo GD estocástico

- Devido à sua natureza estocástica, não apresenta um caminho regular/direto para o mínimo, mudando de direção várias vezes.
- O algoritmo não diminui suavemente até atingir o mínimo, fica "*oscilando*" ou "*ricocheteando*" em torno dele.
- Quando o algoritmo para, os valores finais dos parâmetros são bons, mas não são ótimos.
- A convergência ocorre apenas na média.
- Tempo de treinamento é menor, nesse caso, com apenas uma época o algoritmo já se aproxima do ponto ótimo.
- Necessita de um esquema de ajuste do passo de aprendizagem, α, para ficar mais "*comportado*".

Devido à sua natureza estocástica (ou seja, aleatória), esse algoritmo é muito menos regular do que o gradiente descendente em batelada: em vez de diminuir suavemente até atingir o mínimo, a função de custo irá saltar para cima e para baixo, convergindo apenas na média. Com o passar do tempo, o algoritmo terminará muito próximo do mínimo, mas, quando chegar lá, continuará a ricochetear/oscilar, nunca convergindo (o gradiente estocástico nunca zera definitivamente). Portanto, quando o algoritmo para, os valores finais dos parâmetros são bons, mas não são ótimos.

Quando a função de custo é muito irregular, essa aleatoriedade do algoritmo pode realmente ajuda-lo a escapar de mínimos locais, de modo que o gradiente descendente estocástico tem uma chance maior de encontrar o mínimo global do que o gradiente descendente em batelada.

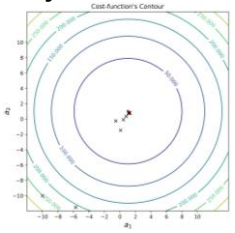
A aleatoriedade do algoritmo é uma faca de dois gumes, pois é boa para escapar de mínimos locais, mas é ruim pois significa que o algoritmo nunca irá se "acomodar" no mínimo global. Uma solução para esse dilema é reduzir gradualmente a taxa de aprendizagem. Os passos começam com grandes valores (o que ajuda a progredir/aprender rapidamente e a escapar de mínimos locais) e depois diminuem cada vez mais, permitindo que o algoritmo se estabilize no mínimo global.

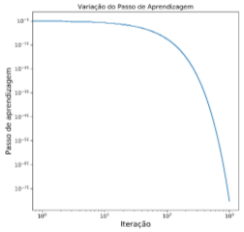
Exemplo: stocastic_gradient_descent_with_figures.ipynb

Slide 43

Implementação: Gradiente Descendente Estocástico com Esquema de Ajuste de α

```
import numpy as np

# Define the number of examples.
M = 1000

# Generate target function.
x1 = np.random.randn(M,1)
x2 = np.random.randn(M,1)
y = x1 + x2 + np.random.randn(M, 1)

# Concatenate both column vectors, x1 and x2.
X = np.c_[x1, x2]

# Number of epochs.
n_epochs = 1

# Initial learning rate.
alpha_int = 0.1
# Learning schedule function.
def learning_schedule(alpha_int, t):
    drop = 0.5
    epochs_drop = 4.0
    alpha = alpha_int * math.pow(drop,  math.floor((1+t)/epochs_drop))
    return alpha

a = np.random.randn(2,1)
# Stocastic gradient-descent loop.
for epoch in range(n_epochs):
  for i in range(M):
    random_index = np.random.randint(M)
    xi = X[random_index:random_index+1]
    yi = y[random_index:random_index+1]
    gradients = -2*xi.T.dot(yi - xi.dot(a))
    alpha = learning_schedule(alpha_int, k, epoch*M + i)
    a = a - alpha * gradients
```





Exemplo GD estocástico

- Exemplo com esquema de variação do passo conhecido como "decaimento gradual"
- O caminho também não é direto/regular para o mínimo.
- Apresenta algumas mudanças de direção ao longo do caminho.
- Oscilação em torno do mínimo é bastante minimizada pelo esquema de variação do passo.
- Os passos começam com grandes valores e depois diminuem cada vez mais, permitindo que o algoritmo se estabilize próximo ao mínimo global.

Os passos começam com grandes valores (o que ajuda a progredir rapidamente e a escapar de mínimos locais) e depois diminuem cada vez mais, permitindo que o algoritmo se estabilize no mínimo global.

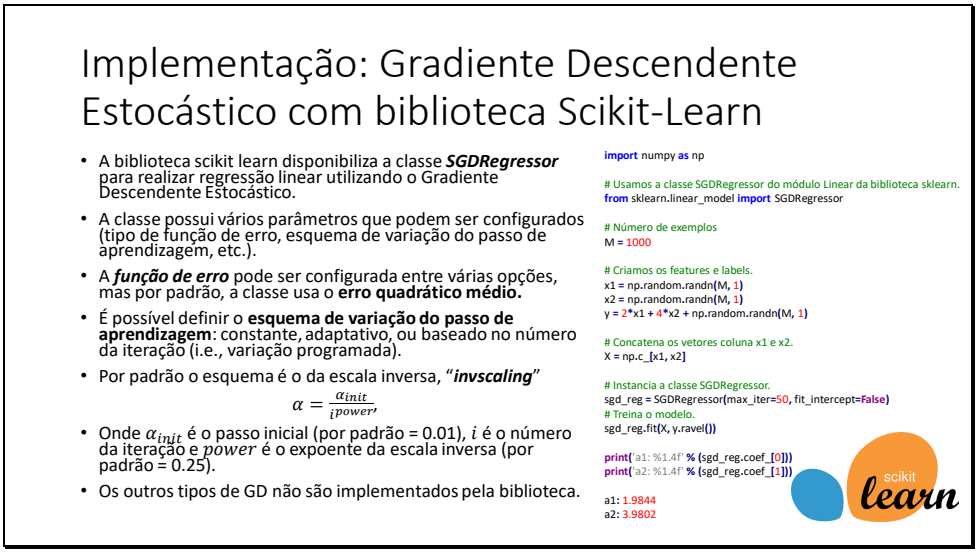
Se a taxa de aprendizagem for reduzida muito rapidamente, o algoritmo poderá ficar preso no mínimo local ou até ficar travado antes de chegar ao mínimo. Se a taxa de aprendizado for reduzida muito lentamente, o algoritmo poderá oscilar ao redor do mínimo por um longo tempo e acabar com uma solução não ótima (sub-ótima) caso o treinamento se encerre muito cedo.

Alguns tipos de esquema para ajuste do passo de aprendizagem são:

- **Decaimento por etapas ou degraus**: reduz a taxa de aprendizado de algum fator a cada número pré-definido de iterações ou épocas. Os valores típicos são utilizados para reduzir a taxa de aprendizado pela metade a cada número pré-definido de épocas. Esses números dependem muito do tipo de problema e do modelo. Uma heurística que você pode ver na prática é observar o erro de validação durante o treinamento com uma taxa de aprendizado fixa e reduzir a taxa de aprendizado em uma constante (por exemplo, 0,5) sempre que o erro de validação parar de decrescer.
- **Decaimento exponencial**: tem a forma matemática α = α0 e^(-kt), onde α0, k são hiperparâmetros e t é o número da iteração (mas você também pode usar o número de épocas).
- **Decaimento temporal**: tem a forma matemática α = α0 / (1 + kt), onde a0, k são hiperparâmetros e t é o número da iteração.

Exemplo: stocastic_gradient_descent_with_learning_schedule_and_with_figures.ipynb

Slide 44



Para executar a regressão linear usando o SGD com o Scikit-Learn, você pode usar a classe SGDRegressor, cujo padrão é otimizar a função de custo do erro ao quadrado. O código a seguir executa 50 épocas, começando com uma taxa de aprendizado de 0,1 (eta0 = 0,1), usando o cronograma de aprendizado padrão (diferente do anterior) e não usa nenhuma regularização (penalidade = Nenhuma;

Informação retirada da documentação da classe SGDRegressor (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html):

learning_rate: string, default='invscaling'
'invscaling': [default]
eta = eta0 / pow(t, power_t)

power_t: double, default=0.25
The exponent for inverse scaling learning rate.

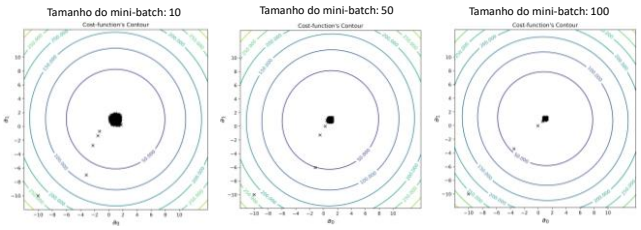
eta0: double, default=0.01
The initial learning rate for the 'constant', 'invscaling' or 'adaptive' schedules. The default value is 0.01.

Slide 45

Implementação: Gradiente Descendente com Mini-Batch

```
import numpy as np

# Define the number of examples.
M = 1000

# Generate target function.
x1 = np.random.randn(M, 1)
x2 = np.random.randn(M, 1)
y = x1 + x2 + np.random.randn(M, 1)

# Add your code here.
```

Exercício da lista #2



- O progresso do algoritmo no espaço de parâmetros é menos irregular do que com o GD estocástico, especialmente com mini-batches grandes o suficiente.
- Como resultado, o mini-batch se aproxima mais do mínimo global do que o GDS.
- Tem comportamento mais próximo do GD em batelada.
- Oscilação em torno do mínimo diminui conforme o tamanho do mini-batch aumenta.
- Pode também ser usado com um esquema de variação do passo de aprendizagem.

O mini-batch é bastante simples de entender quando você conhece o gradiente descendente em batelada e o gradiente descendente estocástico: a cada etapa, em vez de calcular os gradientes com base no conjunto de treinamento completo (como no GD em batelada) ou com base em apenas uma instância (como no GD estocástico), o mini-batch calcula os gradientes em subconjuntos aleatórios de instâncias chamados mini-lotes (do inglês mini-batch).

O progresso do algoritmo no espaço de parâmetros é menos irregular do que com o SGD, especialmente com mini lotes muito grandes. Como resultado, o mini-batch acabará chegando um pouco mais perto do mínimo do que o GDS. Mas, por outro lado, pode ser mais difícil escapar dos mínimos locais (no caso de problemas que sofrem com mínimos locais, diferentemente da Regressão Linear, que como vimos anteriormente apresenta apenas um mínimo, que é o global).

Exemplo: mini_batch_gradient_descent_with_figures.ipynb

Slide 46

Escalonamento de Features

- Em algumas situações, alguns **atributos** acabam sendo dominantes sobre os demais no sentido de que exercerem grande influência sobre o ***erro*** cometido pelo modelo.
- Isto pode ocorrer devido à grande diferença de magnitude entre os atributos.
- Essa diferença de magnitudes afeta o desempenho dos algoritmos de ML.
- **Exemplo**: Suponha que você tenha dois atributos, x_1 e x_2, variando de 0 a 2000 (área de um imóvel em m^2), e de 1 a 5 (número de quartos), respectivamente.
- Algoritmos de ML trabalham com números/magnitudes e não sabem o que eles representam nem suas unidades.
- Porém, algoritmos que usam ***distância*** (e.g., erro quadrático médio) como métrica de erro vão assumir que x_1 tem mais importância do que x_2 e entender que, por exemplo, 100 m^2 > 2 quartos.
- Portanto, o erro entre y e $h(\boldsymbol{x})$ será dominado pelo atributo x_1.

Em geral, os algoritmos de aprendizado de máquina não apresentam bom desempenho quando as features têm escalas muito diferentes.

Por exemplo, muitos algoritmos de ML calculam a distância entre dois pontos pela distância euclidiana. Se um das features tiver uma faixa de valores muito maior do que o de outra feature, o cálculo da distância será regido por essa feature em particular. Portanto, a variação de todos os recursos deve ser escalonada para que cada feature contribua com mesma importância na distância final.

O escalonamento de features é uma técnica para padronizar/normalizar as features em um intervalo fixo. É realizada durante o pré-processamento de dados para lidar com magnitudes, valores ou unidades que tenham grandes variações de valores. Se o escalonamento não for feito, um algoritmo de aprendizado de máquina tende a dar mais importância a valores maiores e dar menos importância a valores menores, independentemente da unidade dos valores.

Por exemplo, se um algoritmo não estiver usando um método de escalonamento, ele poderá considerar o valor de 3000 metros maior que 5 km, mas isso não é verdade e, nesse caso, o algoritmo fornecerá previsões incorretas. Portanto, usamos o escalonamento de features para trazer todos os valores para as mesmas magnitudes e, assim, resolver esse problema.

Os atributos com grandes magnitudes pesam muito mais nos cálculos de distância do que os atributos com pequenas magnitudes.

Intuição:

Algoritmos de aprendizado de máquina funcionam com números e não tem conhecimento do que esses números representam. Um peso de 75 kg e uma distância de 75 quilômetros representam duas coisas completamente diferentes - isso nós, humanos, podemos entender facilmente. Mas para uma máquina, ambos valores 75 são a mesma coisa, independentemente do fato de as unidades de ambos serem diferentes.

Outro exemplo, uma idade média de 30 anos e uma população de 40000 habitantes, são unidades diferentes e portanto 40000 habitantes não pode ser dito ser maior do que 30 anos.

O algoritmo de ML vê apenas números - alguns variando em milhares e outros em torno de dezenas e assume que números maiores tem maior importância. Portanto, valores maiores começam a desempenhar um papel mais decisivo no treinamento do modelo.

É aí que está o problema. A importância da população não é maior do que a importância da idade média, os dois valores não podem ser comparados. Porém, o algoritmo supõe que, desde 54000> 51,7 e 130000> 45,9, e portanto, a população é uma feature mais importante, o que é incorreto.

Esse problema ocorre com todo algoritmo que se baseia no cálculo da distância durante a fase de treinamento.

Escalonamento de atributos/features:

Existem duas maneiras comuns de fazer com que todos os atributos tenham a mesma escala: escalonamento min-max (também conhecido como normalização) e a padronização.
Em alguns casos, ajuda a acelerar a convergência de um algoritmo, como por exemplo, o gradiente descendente.
É aplicado durante pré-processamento dos exemplos de treinamento (i.e., features).

Vantagens:

- Possibilita comparar o peso/influência de cada feature no modelo.
- Melhora o desempenho e a estabilidade do treinamento do modelo.

Escalonamento de Features

- Dada a seguinte equação hipótese, $h(\boldsymbol{x})$

$$\hat{y}(i) = h\big(\boldsymbol{x}(i)\big) = a_1 x_1(i) + a_2 x_2(i).$$

- A função de erro é dada por

$$J_e(\boldsymbol{a}) = \frac{1}{N}\sum_{i=0}^{N-1}\big[y(i) - \big(a_1 x_1(i) + a_2 x_2(i)\big)\big]^2.$$

- Caso $x_1(i) \gg x_2(i), \forall i$, então x_1 tem uma influência maior no erro resultante, o que pode ser expresso de forma aproximada como

$$J_e(\boldsymbol{a}) \approx \frac{1}{N}\sum_{i=0}^{N-1}[y(i) - a_1 x_1(i)]^2.$$

Escalonamento de Features

- Esse problema ocorre com todo algoritmo que se baseia no cálculo da distância durante a fase de treinamento.
- Portanto, para evitar esse problema, a variação de todos os atributos deve ser escalonada para que cada atributo contribua com mesma importância/peso para o cálculo da distância (ou seja, do erro quadrático médio).
- Ou seja, escalona-se os atributos para deixa-los com a mesma faixa de valores/magnitudes.

Slide 49

Escalonamento de Features

- As duas formas mais comuns de escalonamento são:
  - **Normalização Mín-Max**

$$x_k(i) = \frac{x_k(i) - \min(\boldsymbol{x})}{\max(\boldsymbol{x}) - \min(\boldsymbol{x})}$$

  - **Padronização**

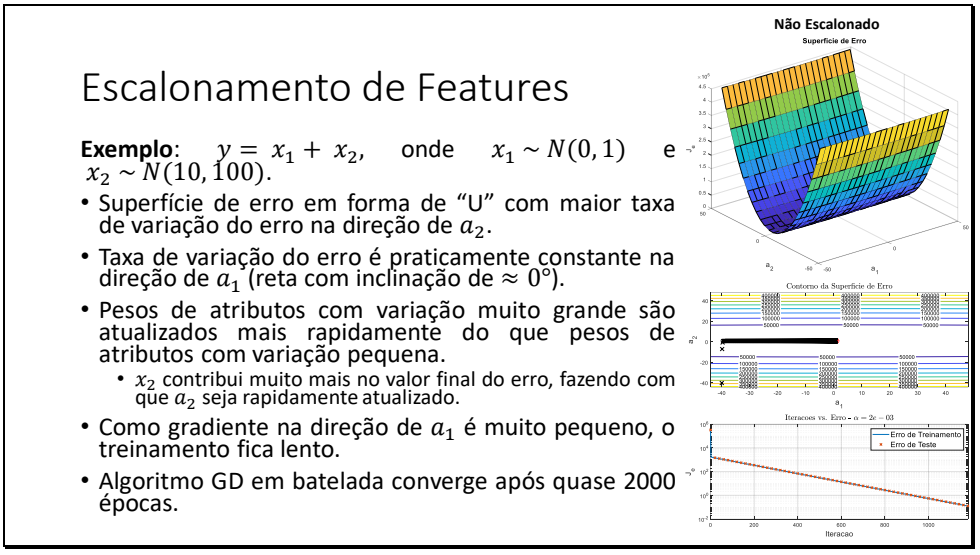
$$x_k(i) = \frac{x_k(i) - \mu_{\boldsymbol{x}}}{\sigma_{\boldsymbol{x}}}$$

- Ajuda a acelerar a convergência do gradiente descendente pois deixa as curvas de nível da superfície de erro mais circulares.
- Possibilita comparar o peso/influência de cada atributo no modelo.
- Aplicado durante pré-processamento das features.
- **OBS.1**: Quando temos um conjunto de validação/teste do modelo, a boa prática é aplicar os valores obtidos durante o escalonamento do conjunto de treinamento ao conjunto de validação.
- **OBS.2**: Em alguns casos, o escalonamento também é aplicado aos rótulos (ou objetivos), i.e., aos valores de y. Mas não se esqueça de desfazer o escalonamento para realizar predições que sejam significativas.

Escalonamento dos objetivos ou rótulos

- https://machinelearningmastery.com/how-to-transform-target-variables-for-regression-with-scikit-learn/
- https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/

Slide 50



Por exemplo, se no caso do gradiente descendente as features tiverem escalas muito diferentes, os parâmetros de features com escala muito grande vão ser atualizados mais rapidamente do que parâmetros de features com escala pequena.

Critério de parada (ou convergência) desse exemplo foi o erro entre épocas/iterações subsequentes cair abaixo de 1e-3 com um máximo número de iterações igual a 10000.

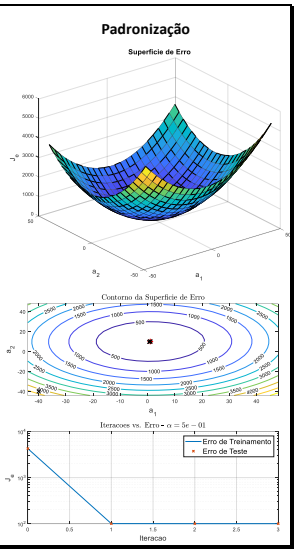
Veja que no caso não-escalonado o algoritmo "converge" com quase 2000 épocas. Nesse caso, a2 é atualizado muito mais rapidamente do que a1, dado que a feature x2 tem variação maior. A variação do gradiente na direção x2 é maior do que na direção x1, ou seja, a descida na direção de x2 é íngreme enquanto na direção de x1 é praticamente uma reta (inclinação igual a 0), fazendo com que a atualização de a1 seja muito pequena.

O aumento da variação de uma das features faz com que o círculos de contorno se tornem elipses que tendem a linhas paralelas quando essa variação é muito grande em relação a outra feature. Denotando que uma das features tem variação muito maior do que a da outra. Outra forma de ver isso, é notar que como x2 tem variação maior do que x1, o erro ao longo de a2 varia muito mais rapidamente do que ao longo de a1, mostrando que x2 contribui muito mais no valor final do erro e que x1 tem pouca contribuição no valor do erro.

Slide 51

Escalonamento de Features

- Agora aplicamos ***padronização*** aos atributos.
- A superfície se aproxima mais da forma de uma "*tigela*".
- Círculos de contorno mais "circulares", denotando que a superfície tem inclinação similar em todas as direções dado que os atributos agora tem variações similares.
- Algoritmo converge em apenas 3 épocas.
- Treinamento é rápido pois a inclinação é íngreme em todas as direções.



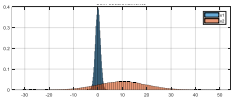
Já no caso padronizado, a superfície se torna mais próxima de uma tigela, com círculos de contorno mais "circulares", denotando que a superfície tem inclinação similar em todas as direções pois os atributos agora tem variações similares. Desta forma, vemos que ambas features, x1 e x2, contribuem igualmente para o cálculo do erro.

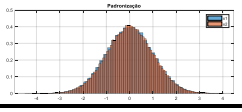
Nota-se também que algoritmo converge em apenas 3 épocas.

Slide 52

Escalonamento de Features com SciKit-Learn

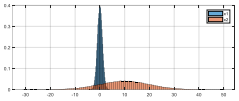
```
# Import Class StandardScaler from module Preprocessing of library sklearn
responsible for standardizing the data.
from sklearn.preprocessing import StandardScaler

# Instantiate a Standard scaler.
stdScaler = StandardScaler()

# Concatenate both column vectors.
X = np.c_[x1, x2]

# Standardize the features.
scaled_X = stdScaler.fit_transform(X)
```

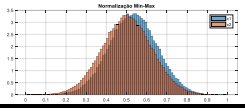




Exemplo: escalonamento

```
# Import Class MinMaxScaler from module Preprocessing of library sklearn
responsible for normalizing the data.
from sklearn.preprocessing import MinMaxScaler

# Instantiate a MinMax scaler.
minMaxScaler = MinMaxScaler()

# Concatenate both column vectors.
X = np.c_[x1, x2]

# Standardize the features.
scaled_X = minMaxScaler.fit_transform(X)
```





Exemplo: escalonamento_de_atributos_com_scikit_learn.ipynb

Slide 53

Tarefas e Avisos

- Exemplos vão estar disponíveis no site.
- Lista #2 vai estar no site ainda hoje e já pode ser resolvida.
- Lista #0 e #1 deve ser entregue até hoje à meia-noite.
- Prazo para definição do tema do projeto final foi até 24/03 (14 grupos).
- Atendimento às quartas-feiras das 8 as 10 da manhã (Skype: zz4fap)
- https://www.inatel.br/docentes/felipefigueiredo/