Universidade Estadual de Campinas

Instituto de Matemática, Estatística e Computação Científica

Machine Learning

Projeto I: Regressão Linear e Regressão Logística

Davi Wanderley Misturini (RA: 235799)

Professor Dr. Joao Batista Florindo

Setembro de 2021

Sumário

Capítulo 1

Introdução

Técnicas de *Aprendizado de Maquinas* tem sido aplicadas em uma variedade de problemas na ciência e na indústria durante os últimos anos, como em casos de detecção de tumores, carros autônomos, processamento de linguagens naturais, predição do valor de ações, entre outros. É salutar, desta forma, prender as técnicas e implementações básicas dos algoritmos clássicos dispostos na literatura atual.

Para a primeira parte desse projeto, iremos aplicar os métodos de regressão linear para ajustar a evolução do número de casos dos primeiros 140 dias da COVID-19 no Brasil. Ainda para o caso da regressão linear, serão feitos os ajustes polinomiais e exponencial. Número de iterações e taxa de aprendizagem serão variados para um melhor estudo do caso. Nosso interesse será nos valores da função de custo, que será nosso valor de quantificação da eficácia de nosso método.

Na segunda atividade, utilizaremos a regressão logística para solucionar um problema de classificação. Nesse caso, nosso maior interesse será na porcentagem de acerto na classificação das imagens do conhecido banco de dados MNIST, em que são apresentadas diversas amostras de imagens de números de 0 a 10 escritas à mão, de tamanho 20×20, totalizando 5000 exemplos, separadas convenientemente para treino e teste.

Capítulo 2

Parte I - Regressão Linear

Analisaremos os dados do arquivo *casesBrazil.csv*, que contém os números oficiais de casos de COVID-19 no Brasil, a partir de 25 fevereiro de 2020 até os 100 dias seguintes.

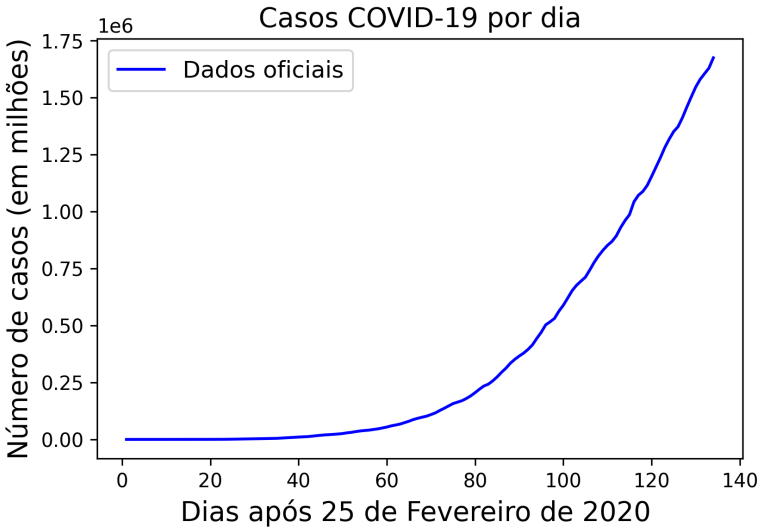


Figura 2.1: Gráfico da dispersão de dados dos casos de COVID-19 no Brasil.

2.1 Item 1

Como vimos, a regressão linear pode ser usada também para o ajuste de curvas não lineares, bastando que se façam as transformações adequadas dos atributos. Com base nisso, implemente um algoritmo de regressão linear que ajuste um polinômio de grau n à curva do número de casos, usando **gradiente descendente** para a obtenção dos parâmetros. Faça testes com $n = 3, 5, 10$ e desenhe a curva de cada polinômio sobre os dados originais.

Com alguns ajustes ao algorítimo de Regressão Linear Multivariada (RLM), é possível utilizar regressão linear sem que a curva definida pelos dados sejam necessariamente lineares. Para isto, devemos adicionar atributos não lineares à função de hipótese. Neste caso, normalizar os dados é ainda mais necessário.

11 errors4 warnings

Usaremos as seguintes notações:

$\{(x_i, y_i)\}$: Conjunto de treinamento

(x_i, y_i): i-ésimo exemplo de treinamento

$x_j^{(i)}$: j-ésimo atributo do i-ésimo exemplo de treinamento

m: Número de exemplos de treinamento

n: Número de atributos

x: Variáveis de entrada / atributos (*features*)

y: Variável de saída / alvo (*target*)

Seja $h_\theta(x)$ a função de hipótese dada por

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_n x^n \tag{2.1}$$

Tomando $x \in \mathbb{R}^n$ na forma de vetor coluna, isto é,

$$x = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ x^{(3)} \\ \vdots \\ x^{(m)} \end{bmatrix}_{m \times 1}$$

Para determinar os parâmetros de $y = h_\theta(x)$, podemos resolver a equação matricial

$$X\theta = y \tag{2.2}$$

onde X é a matriz de design dada por

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} & \cdots & x_n^{(3)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}_{m \times (n+1)}$$

o vetor de saída y é dado por

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \vdots \\ y^{(m)} \end{bmatrix}_{m \times 1}$$

e θ é denotado matricialmente como

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}_{(n+1) \times 1}$$

Na regressão linear, o objeto é minimizar a função de custo $J(\theta)$, definida pelo erro quadrático médio:

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^{m} (h_\theta(x_i) - y_i)^2 \tag{2.3}$$

Para tal, podemos utilizar o método do gradiente descendente, repetindo o algoritmo até que

$$\theta_j \coloneqq \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{2.4}$$

convirja. É necessário um cuidado particular na escolha da taxa de aprendizado α: se muito grande, $J(\theta)$ pode não convergir. Se muito pequeno, pode demorar muito para chegar no ponto que minimize $J(\theta)$.

Definido os objetos básicos, o próximo passo é calcular $\frac{\partial}{\partial\theta_j}J(\theta)$. Pela regra da cadeia, temos

$$\begin{aligned}\frac{\partial}{\partial\theta_j}J(\theta) &= \frac{\partial}{\partial\theta_j}\left[\frac{1}{2m}\sum_{i=0}^{m}(h_\theta(x_i)-y_i)^2\right] &(2.5)\\ &= \frac{1}{2m}\cdot 2\left[\sum_{i=0}^{m}(h_\theta(x_i)-y_i)\right]\cdot\frac{\partial}{\partial\theta_j}(h_\theta(x_i)) &(2.6)\\ &= \frac{1}{m}\left[\sum_{i=0}^{m}(h_\theta(x_i)-y_i)\right]\cdot\frac{\partial}{\partial\theta_j}(h_\theta(x_i)) &(2.7)\end{aligned}$$

De (2.1), temos, para $j \leq n$,

$$\begin{aligned}\frac{\partial}{\partial\theta_j}(h_\theta(x_i)) &= \frac{\partial}{\partial\theta_j}(\theta_0+\theta_1x_i+\theta_2(x_i)^2+\cdots+\theta_n(x_i)^n) &(2.8)\\ &= (x_i)^j &(2.9)\end{aligned}$$

Substituindo (2.9) em (2.7), segue que

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=0}^{m}(h_\theta(x_i)-y_i)x_i^j \quad (2.10)$$

Matricialmente,

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}(X\theta-y)e_i^Tx_i^j \quad (2.11)$$

onde e_i^T denota a transposta do vetor coluna canônico

$$e_j = \begin{bmatrix}0\\ \vdots\\ 1\\ \vdots\\ 0\end{bmatrix}_{(m+1)\times 1}$$

(assume 1 na entrada j e 0 nas demais). Portanto, substituindo (2.15) em (2.4), devemos repetir

$$\theta_j \coloneqq \theta_j - \alpha\frac{1}{m}(X\theta-y)e_i^Tx_i^j \quad (2.12)$$

até a convergência.

Agora, analisaremos os casos em que a função de hipótese é um polinômio de grau 3, 5 e 10, respectivamente. Para tal, normalizaremos os nossos dados. Através do gráfico

QQ (Figura 2.2) notamos que os dados não seguem uma distribuição normal. Desta forma, utilamos o método de *scaling* para normalização, que consiste em atualizar os exemplos de treinamento (x_j, y_j) de forma que

$$x_j := \frac{x_j - \min(x_j)}{\max(x_j) - \min(x_j)} \tag{2.13}$$

$$y_j := \frac{y_j - \min(y_j)}{\max(y_j) - \min(y_j)} \tag{2.14}$$

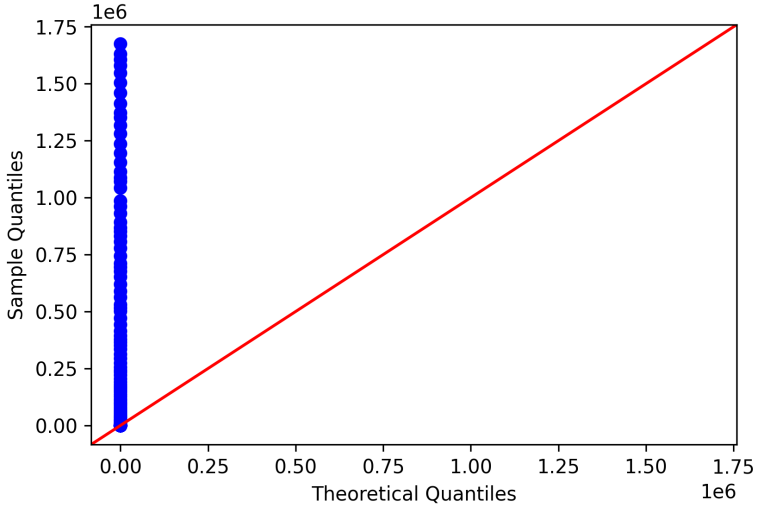
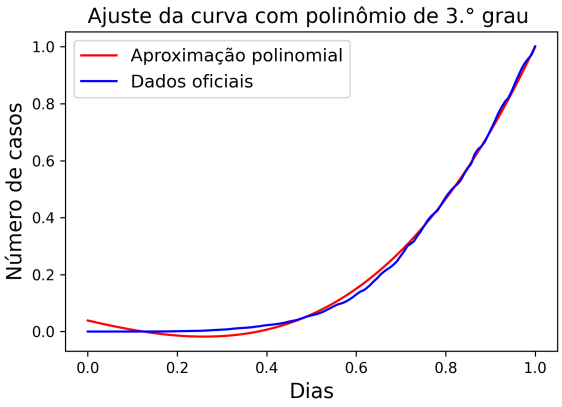


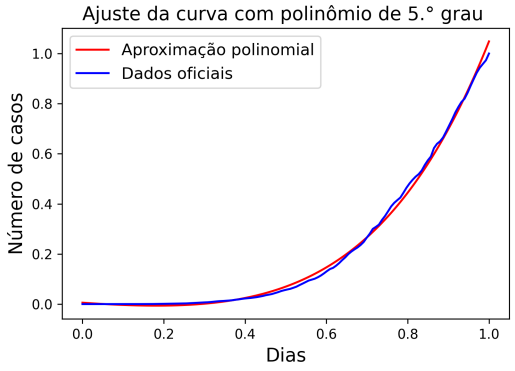
Figura 2.2: No Gráfico QQ, os dados não se ajustam à reta. Desta forma, a distribuição não é normal.

A Figura 2.3 mostra o ajuste de curva para o caso dos polinômios de grau 3, 4 e 5.

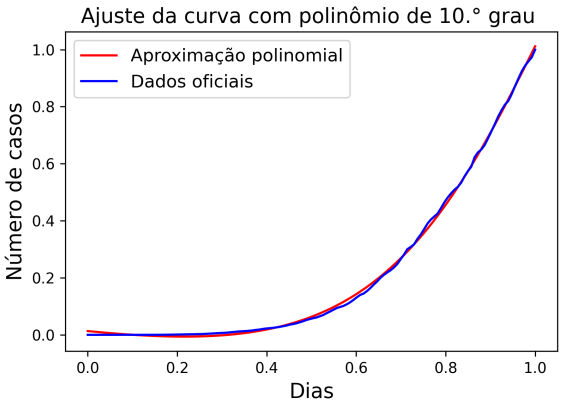
Observe que, a medida que aumentamos o grau do polinômio, o ajuste entre as curvas é melhorado (ao menos visualmente). A Tabela 2.1 mostra os thetas e valores da função de custo encontrados, em cada caso, com 1000 iterações.



(a)



(b)



(c)

Figura 2.3: Comparação entre o ajuste de curvas.

Grau do Polinômio	Theta	Valor da função de custo
3	[0.038905, -0.367021, 0.307818, 1.021825]	0.032859
5	[0.005718, -0.115295, 0.198506, 0.334639, 0.340692, 0.284131]	0.022346
10	[0.0132941, -0.155247, 0.206934, 0.347884, 0.340376, 0.265645, 0.169131, 0.072829 -0.013777, -0.087461, -0.147961]	0.008754

Tabela 2.1: Comparação entre diferentes tetas e valor da função de custo de acordo com o grau do polinômio, com 1000 iterações.

2.2 Item 2

A inspeção visual dos dados mostra que a curva poderia ser melhor aproximada por uma curva exponencial, ou seja:

$$\theta_0 e^{\theta_1 x}$$

Implementamos um algoritmo de regressão linear com gradiente descendente e obtemos valores θ_1 e θ_2. Mostre a curva exponencial obtida sobreposta aos dados originais. DICA: Em vez de aproximar y por $h_\theta(x)$, aproxime $\log y$ por $\log h_\theta(x)$.

Seja a seguinte função de hipótese:

$$h_\theta(x) = \theta_0 e^{\theta_1 x}. \tag{2.15}$$

Aplicando log em ambos os lados da igualdade em (2.15), obtemos

$$\log(h_\theta(x)) = \log(\theta_0) + \theta_1 x. \tag{2.16}$$

Agora, definindo

$$b = \log(\theta_0) \tag{2.17}$$

$$Y = \log(h_\theta(x)) \tag{2.18}$$

podemos escrever

$$Y = b + \theta_1 x \tag{2.19}$$

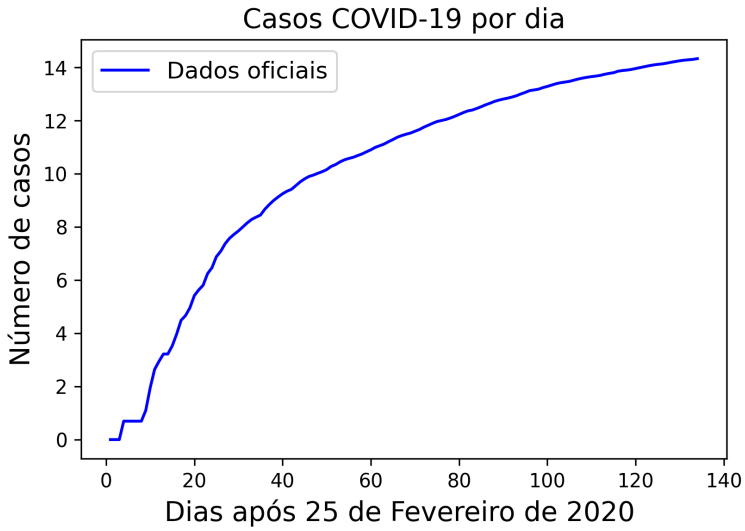


Figura 2.4: Dados de COVID-19 reescalados via função log.

A Tabela 2.2 mostra o resultado obtido com normalização dos dados e 1000 iterações. A Figura 2.5 mostra o ajuste da curva por meio da função exponencial.

Thetas	Valor da função de custo
$[-5.381989, 5.565047]$	0.249007

Tabela 2.2: Thetas e função de custo encontrados, após 1000 iterações

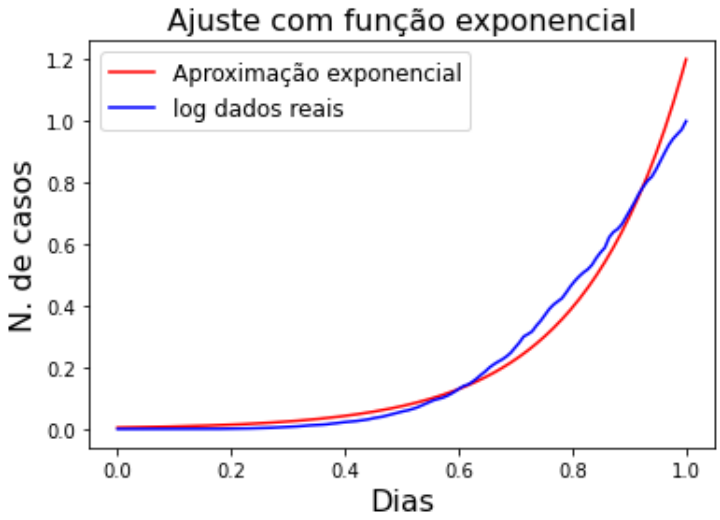


Figura 2.5: Ajuste da curva exponencial com o log dos dados reais

Já na Figura 2.6, observamos o ajuste na reta de regressão.

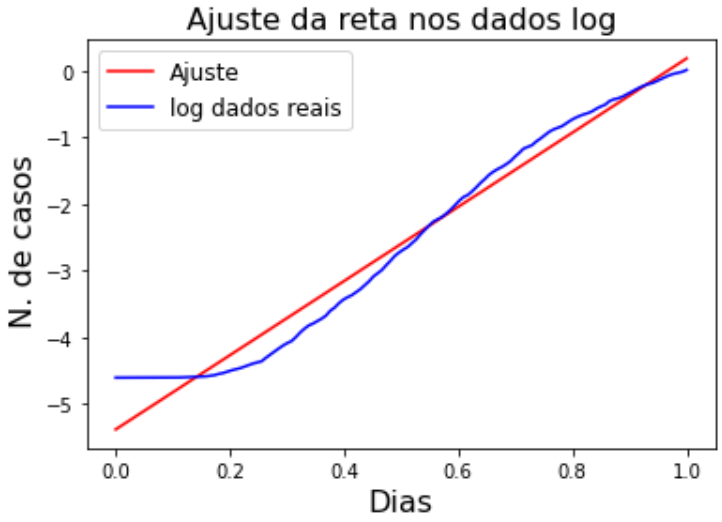
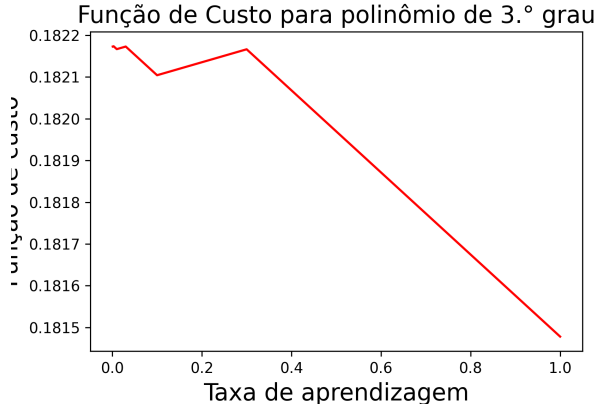


Figura 2.6: Ajuste da curva exponencial com o log dos dados reais

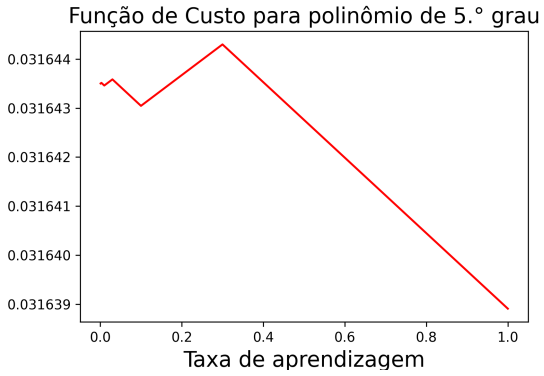
2.3 Item 3

Teste diferentes valores para a taxa de aprendizado α e mostre um gráfico do custo J em função de α. SUGESTÃO: Usar $\alpha = 10, 3, 1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001$.

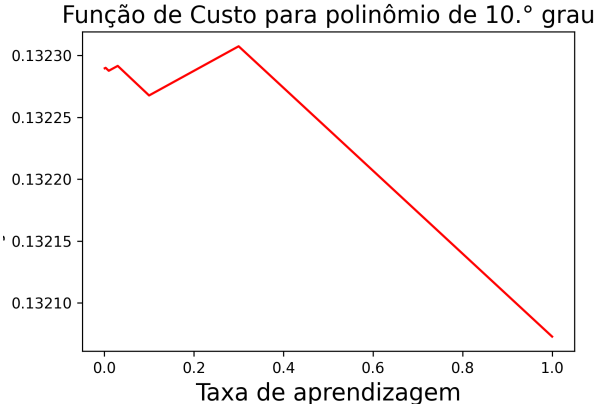
Agora, testamos diferentes valores para a taxa de aprendizado. Observou-se que a função de custo é minimizada em $\alpha = 1$, seja qual dos três polinômios for analisado, conforme mostra a Figura 2.7 e a Tabela 2.3, a qual exibe os valores de α com os respectivos custos. Optamos por omitir nos gráficos o caso em que $\alpha = 10$. Com essa taxa de aprendizagem muito elevada, o valor da função de custo da um "salto", o que dificultaria a análise visual dos dados.



(a)



(b)



(c)

Figura 2.7: Valores da função de custo para cada α e polinômio escolhido.

Polinômio de grau 3

α	**custo**
0.001	0.182172
0.003	0.182173
0.01	0.182166
0.03	0.182172
0.1	0.182104
0.3	0.182166
0.1	0.181478
10	1.475740

Polinômio de grau 5

α	**custo**
0.001	0.031643
0.003	0.031643
0.01	0.031643
0.03	0.031643
0.1	0.031643
0.3	0.031644
0.1	0.031638
10	4.416259

Polinômio de grau 10

α	**custo**
0.001	0.132289
0.003	0.132289
0.01	0.132287
0.03	0.132291
0.1	0.132267
0.3	0.132307
0.1	0.132072
10	8.197385

Tabela 2.3: Valores da função de custo (**custo**) para diferentes α, de acordo com o grau do polinômio.

A mesma análise foi feita para o caso da curva exponencial (Figura 2.8 e Tabela 2.4). É interessante observar que no caso em que $\alpha = 10$, tivemos *inf* como output. Isso significa que não houve convergência com tal taxa de aprendizagem.

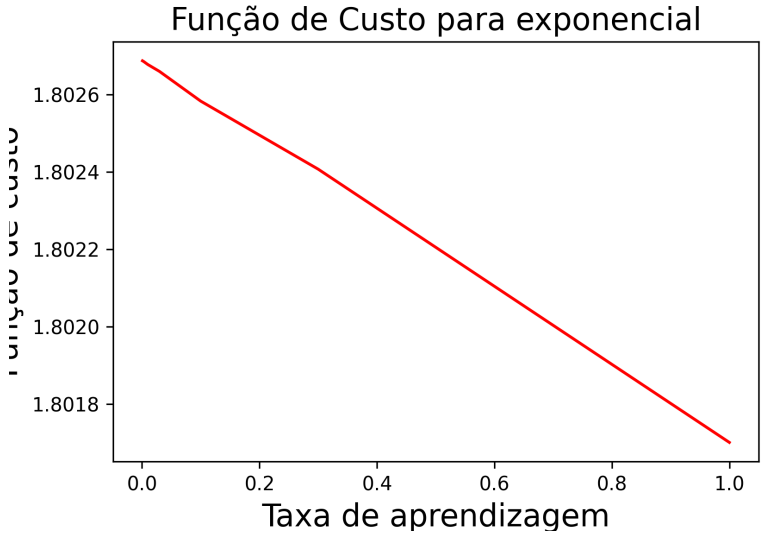


Figura 2.8: Gráfico da função de custo no caso exponencial, para cada α proposto.

Função exponencial

α	**custo**
0.001	1.802687
0.003	1.802685
0.01	1.802677
0.03	1.802659
0.1	1.802583
0.3	1.802407
0.1	1.801701
10	*inf*

Tabela 2.4: Valores da função de custo (**custo**) para diferentes α, no caso exponencial.

2.4 Item 4

Implemente o ajuste de $\theta_0 e^{\theta_1 x}$, mas agora usando equações normais.

Em termos matemáticos, queremos minimizar a equação 2.3 em termos de θ. Assim, podemos ignorar o termo independente $1/2m$ e considerar

$$J(\theta) = \sum_{i=0}^{m} (h_\theta(x_i) - y_i)^2 = (\theta^T X - y)(\theta^T X - y)^T, \tag{2.20}$$

em que $\theta : n \times 1$, $X : n \times m$, $y : 1 \times m$. Idealmente, queremos encontrar θ de forma que

$$\frac{\partial J(\theta)}{\partial \theta} = 0. \tag{2.21}$$

Reescrevendo (2.20), temos

$$J(\theta) = \theta^T X X^T \theta - \theta^T X y^T - y X^T \theta + y y^t \tag{2.22}$$

já que $\theta^T X y^T = y X^T \theta$. Desta forma,

$$\frac{\partial J(\theta)}{\partial \theta} = 2 X X^T \theta - 2 X y^T = 0 \tag{2.23}$$

Portanto, sendo XX^T não singular, temos

$$\theta = (X X^T)^{-1} X y^t. \tag{2.24}$$

Após implementação, obtemos os resultados seguintes, mostrados na Figura 2.9 e Tabela 2.5

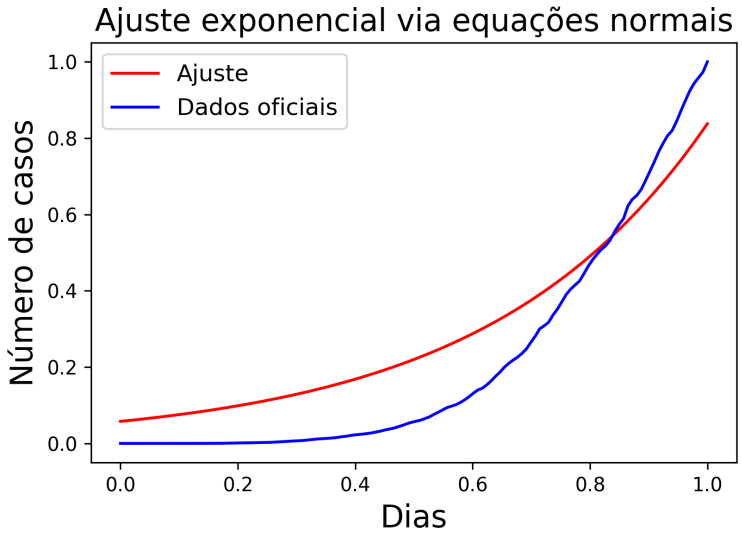


Figura 2.9: Gráfico do ajuste de curva no caso exponencial via equações normais

Thetas	Valor da função de custo
$[0.057768, 2.674307]$	1.797958

Tabela 2.5: Valores dos thetas e da função de custo encontrados, no caso de ajuste exponencial com equações normais

Observe que houve um avanço na diminuição do valor da função de custo no caso em que usamos equações normais, em comparação com a Tabela 2.2, em que não utilizamos tal abordagem.

Capítulo 3

Parte II - Regressão Logística

Nesta parte da atividade, utilizaremos regressão logística para reconhecimento de dígitos manuscritos, por meio da base de imagens MNIST.

3.1 Item 1

Implemente a regressão logística multi-classes para resolver este problema. Use vetorização sempre que possível. Mostre a porcentagem de imagens classificadas corretamente. Mostre também as imagens que ele classificou incorretamente, juntamente com o dígito verdadeiro e o dígito que o classificador atribuiu.

No caso da regressão logística, definimos a função de hipótese $h_\theta(x)$ como

$$h_\theta(x) = g(\theta^T x) \tag{3.1}$$

onde g é a função sigmoide dada por

$$g(z) = \frac{1}{1 + e^{-z}}. \tag{3.2}$$

Pelo princípio da verossimilhança, obtêm-se a função de custo $J(\theta)$ definida como

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \Big[y^{(i)} \log(h_\theta x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta x^{(i)}) \Big]. \tag{3.3}$$

O algoritmo do gradiente descendente é idêntico ao da regressão linear, com o detalhe para a nova função de hipótese definida em (3.2). Isto é,

$$\theta_j \coloneqq \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{3.4}$$

Recordando que, neste caso, a normalização dos dados é ainda mais necessária. Além disso, podemos reescrever (3.4) na forma vetorizada:

$$\theta_j := \theta_j - \frac{\alpha}{m} X^T(g(X\theta) - y). \tag{3.5}$$

No caso concreto, separamos 80% das imagens da MNIST para treino e 20% para teste, obtendo uma acurácia de 90.4%. A matriz de confusão (Figura 3.1) ajuda a entendermos melhor a performance deste algoritmo de classificação.

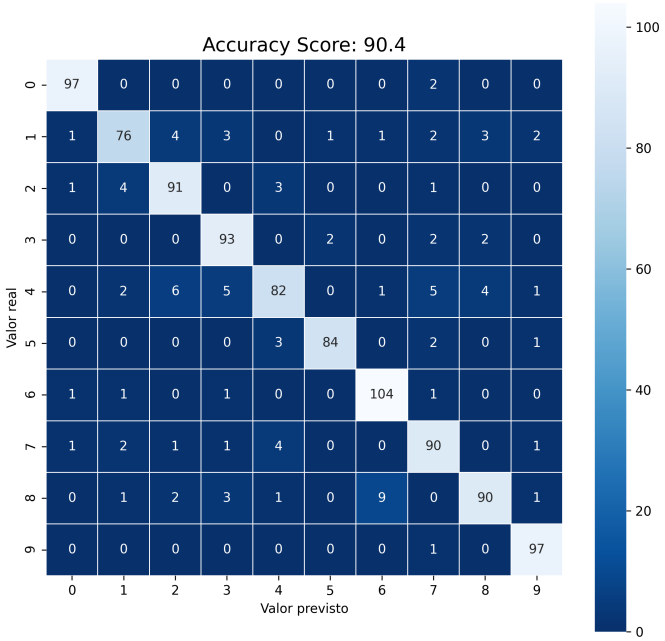


Figura 3.1: Matriz de confusão nos dados da MNIST com regressão logística multiclasse

Seja $A : 10 \times 10$ a matriz de confusão representada na Figura 3.1. Os elementos da diagonal $A(i, i) = k_{i-1}$ (fixado i inteiro entre 1 e 10) indicam que o algoritmo previu corretamente k_{i-1} imagens da classe de números $i - 1$. Por exemplo, $A(1, 1) = 97$ indica que 97 imagens foram classificadas corretamente como da classe numérica 0. Já o elemento $A(2, 3) = 4$, por exemplo, indica que o algoritmo classificou erroneamente 4 imagens como classe de números 2, quando na verdade deveria ser classificada como da classe de números 1.

3.2 Item 2

Implemente agora a regressão logística multi-classes regularizada para o mesmo problema. Mostre a porcentagem de acerto e as imagens classificadas erroneamente.

Adicionando o termo de regularização, a função de custo toma a forma

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta x^{(i)}) + (1-y^{(i)})\log(1-h_\theta x^{(i)})\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2. \tag{3.6}$$

e o gradiente descendente é dado por

$$\theta_0 \coloneqq \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)} \tag{3.7}$$

$$\theta_j \coloneqq \theta_j - \frac{\alpha}{m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \lambda\theta_j\right] \tag{3.8}$$

onde $j = 1, \ldots, n$.

Novamente, utilizamos a regra 80/20 (80% dos dados para treino e 20% para teste) com $\lambda = 0.1$. Obtivemos uma acurácia de 90.6. É interessante notar que o resultado obtido foi muito similar ao da forma não regularizada, o que significa que o primeiro caso (Item 1) já possuía grau suficiente de generalização (baixo overfitting). A matriz de confusão está representada na Figura 3.2, cuja interpretação é análoga ao caso anterior.

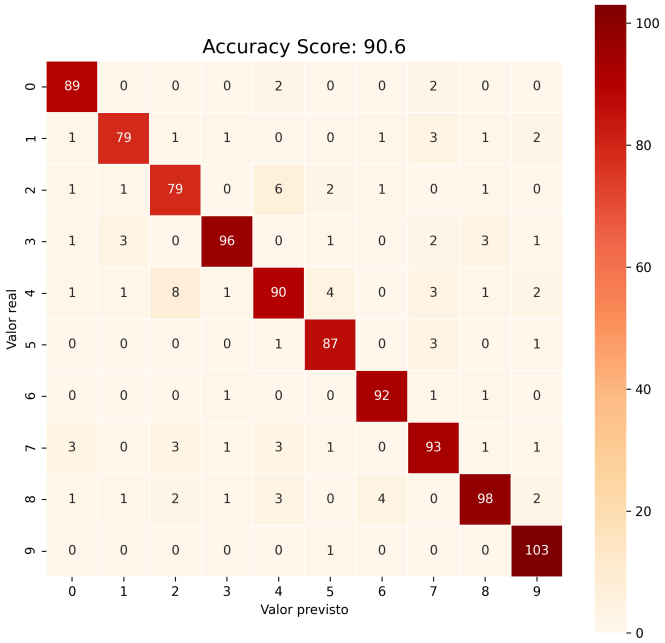


Figura 3.2: Matriz de confusão nos dados da MNIST com regressão logística multi-classe regularizada