UNIVERSIDADE FEDERAL DO PARANÁ

ROGÉRIO OTAVIO MAINARDES DA SILVA

MÉTODO DO GRADIENTE ESTOCÁSTICO PARA OTIMIZAÇÃO DE FUNÇÕES SEPARÁVEIS EM APRENDIZAGEM DE MÁQUINA

CURITIBA
2021

ROGÉRIO OTAVIO MAINARDES DA SILVA

MÉTODO DO GRADIENTE ESTOCÁSTICO PARA OTIMIZAÇÃO DE FUNÇÕES SEPARÁVEIS EM APRENDIZAGEM DE MÁQUINA

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Licenciado, Curso de Matemática, Departamento de Matemática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Dr. Abel Soares Siqueira

CURITIBA
2021

RESUMO

Neste trabalho estudamos e implementamos os algoritmos do Gradiente Descendente e Gradiente Estocástico com *mini-batch*, algoritmos clássicos do contexto da Aprendizagem de Máquina implementados aqui em um pacote de linguagem Julia para a resolução de problemas de Regressão Linear e Logística. Apresentamos os conceitos e definições básicas da Aprendizagem de Máquina e a formulação dos problemas de Regressão Linear e Logística. Para a resolução destes problemas, são introduzidos conceitos de otimização e a construção dos algoritmos com seus devidos embasamentos teóricos. Além disso, são apontados os detalhes e particularidades da implementação do algoritmo, com suas motivações e testes de funcionalidade.

palavras-chave: aprendizagem de máquina, gradiente descendente, gradiente estocástico, regressão linear, regressão logística.

Sumário

1 Introdução **5**

2 Aprendizagem de Máquina **6**

2.1 Regressão e Quadrados Mínimos 11

2.2 Classificação com Regressão Logística 15

2.3 Redes Neurais - O Perceptron Multicamadas 19

2.4 Padronização dos problemas 22

3 Otimização **24**

3.1 Método do gradiente e sua convergência 25

3.2 O problema do Método do Gradiente Descendente 29

3.3 Gradiente estocástico e sua convergência 30

3.4 O problema do gradiente estocástico 32

3.5 Gradiente estocástico *mini-batch* 33

3.6 Regularização 35

4 Implementação e Resultados **39**

4.1 O pacote MultObjNLPModels 39

4.2 A estrutura RegressionModels 40

4.3 Implementação do gradiente estocástico 41

4.4 Comparações 48

4.5 Implementação da Rede Neural com RegressionModels 58

4.6 Comparação com o pacote *scikit-learn* 62

5 Considerações finais **66**

Referências **67**

1 Introdução

A cada dia que passa é possível notar cada vez mais o acelerado avanço da tecnologia. Em meio ao desenvolvimento de novos *gadgets*, programas computacionais e facilidades, o estudo da Inteligência Artificial tem se tornado cada vez mais presente tanto na academia quanto nas pequenas e grandes empresas, sempre visando utilizá-la de modo a tirar proveito desta tecnologia da melhor forma.

Dentro desta gigante gama de estudos da Inteligência Artificial surge o campo da Aprendizagem de Máquina, que consiste no desenvolvimento de modelos e funções matemáticas de modo a ajustá-las em conjuntos de dados para que então seja possível realizar previsões e inferências identificando determinados padrões destes conjuntos, auxiliando muito no processo da tomada de decisões.

Tendo em vista este cenário, este trabalho vem para colocar em prática os estudos destes conceitos de Aprendizagem de Máquina. Atualmente, muitas bibliotecas de determinadas linguagens de programação têm algoritmos já desenvolvidos e prontos para a aplicação. A proposta aqui trata de desenvolver um algoritmo de Aprendizagem de Máquina na linguagem de programação Julia para realizar comparações com estes já existentes.

Sendo assim, para o desenvolvimento deste trabalho, inicia-se com um estudo dos conceitos de Aprendizagem de Máquina, apresentando alguns exemplos, definições, nomenclaturas e sua estrutura básica. Seguido então da explicação dos problemas sob os quais o algoritmo foi desenvolvido para solucionar: Regressão Linear e Regressão Logística.

Passa-se então para uma abordagem da Otimização Matemática, para a apresentação dos lemas e teoremas que garantem as convergências dos modelos. Com isso realizado, são descritos os algoritmos e seus funcionamentos.

A partir daí, são apontados os prós e contras de cada método construído: Grandiente Descendente e Estocástico, seguido do que seria o intermediário dentre estes dois, o Gradiente Estocástico *mini-batch*, para finalmente apresentar as peculiaridades do modelo como as taxas de aprendizagem e a necessidade da Regularização.

Finalizamos o trabalho então com os resultados obtidos com o algoritmo, analisando seu desempenho, suas variações de parâmetros e possibilidades de aplicação, além da comparação com uma das bibliotecas de programação mais famosa da atualidade, o *scikit-learn*.

O código dos métodos desenvolvidos neste trabalho pode ser encontrado no repositório em *Github* do pacote *MultObjNLPModels* no link `https://github.com/CiDAMO/MultObjNLPModels.jl`. E o repositório dos testes realizados em `https://github.com/RogerioOMDS/TCC-tests`.

2 Aprendizagem de Máquina

A Aprendizagem de Máquina é um subcampo da Inteligência Artificial que consiste no ajuste de funções matemáticas a bancos de dados, identificando seus padrões a partir de determinados algoritmos computacionais.

Realizando este ajuste ao banco de dados de maneira adequada, torna-se possível fazer suposições futuras, obter novas informações e gerar novas hipóteses sobre tais dados. Estas informações podem auxiliar na tomada de decisões em áreas industriais, empresariais, em diferentes ramos da saúde e muitas outras.

Um exemplo de serviços empresarial que utiliza da Aprendizagem de Máquina é a aplicação de algoritmos que tentam prever as oscilações da bolsa de valores, aprendendo com padrões passados destes gráficos e tentando reproduzi-los futuramente para obter informações de quando vender ou comprar determinadas ações.

Já no contexto hospitalar e de saúde, existem muitos estudos sobre a identificação de células cancerígenas a partir do processamento de imagens, ou seja, ensina-se o computador a identificar quais células podem ou não ser cancerígenas e com isso, pode-se atribuir algum diagnóstico com muito mais velocidade ao paciente.

Outra aplicação da Aprendizagem de Máquina ocorre nos *chatbots*, robôs que conversam e interagem digitalmente com humanos em atendimentos e encaminhamentos prévios em determinadas empresas, tornando o serviço mais dinâmico.

Alguns outros exemplos destes estudos estão mostrados a seguir. Começando com um banco de dados muito clássico para problemas de classificação: o *Iris Data Set* [5] consiste em um conjunto de dados que contém medidas de comprimento e largura de pétalas e sépalas de determinadas flores e tem como objetivo, dada uma planta nova da mesma espécie, classificá-la entre *Iris Setosa, Iris Versicolor ou Iris Virgínica*. A seguir é possível conferir o diagrama de dispersão que considera *Comprimento* $\times$ *Largura* das pétalas após a classificação das mesmas. Com o diagrama da Figura 1 é possível perceber que flores da mesma classe tendem a se agrupar em regiões semelhantes.

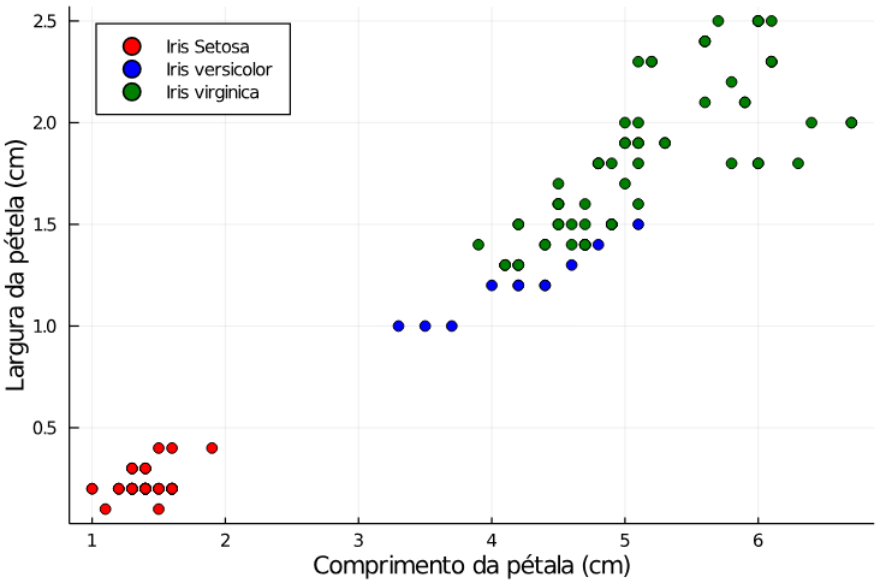


Figura 1: Classificação de flores a partir de suas medidas.

Como um exemplo mais complexo temos o chamado *deep learning*, que para a aprendizagem utiliza um modelo matemático mais complexo que recebe o nome de Rede Neural Multicamada. Neste exemplo, o banco de dados consiste em caracteres matemáticos alfa numéricos escritos a mão de diversas formas. O objetivo aqui consiste em ensinar o modelo a identificar caracteres matemáticos escritos a mão utilizando uma câmera para o reconhecimento instantâneo.

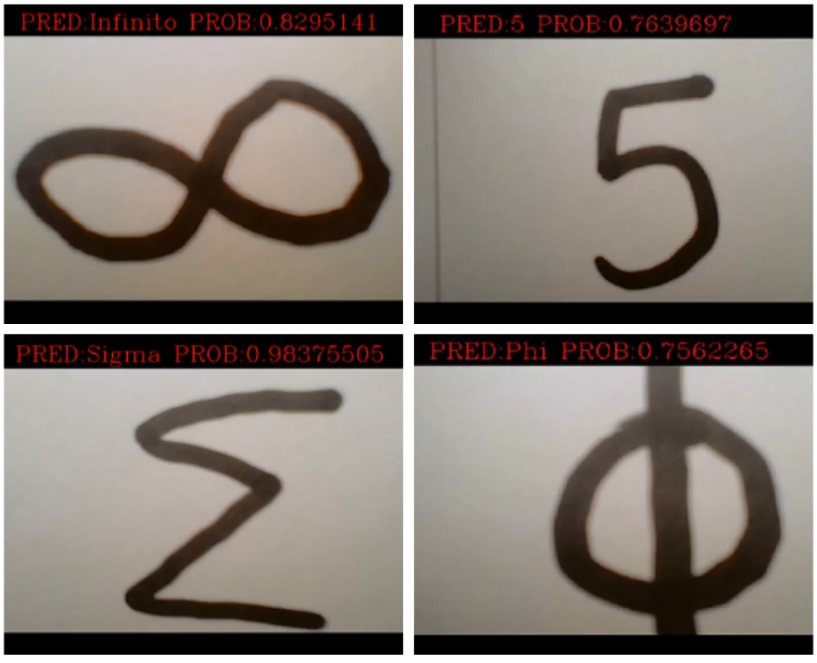


Figura 2: Predição de Caracteres Matemáticos escritos a mão.

Para a retirada de tais conhecimentos de um conjunto de dados, ou seja, para este

processo de ensino da máquina, existem alguns requisitos a serem cumpridos:

1. **Banco de dados**
   Como apontado já anteriormente, a aprendizagem de máquina é realizada sobre um banco de dados. Este deve ser tratado para formar um conjunto consistente para o treinamento do modelo matemático.

2. **Conjunto de dados para treinamento**
   Este surge a partir do item anterior. Com o banco de dados já tratado, este tem uma parcela separada apenas para o modelo matemático se ajustar e descrever melhor tais dados.

3. **Conjunto de dados para teste**
   Também a partir do tratamento citado no primeiro item. Este se constitui da parcela dos dados que não é utilizada no treino no modelo. Este conjunto, comumente menor do que o conjunto de treinamento, é separado apenas para avaliar a qualidade do ajuste do modelo nos dados de treino.

4. **Modelo matemático**
   O modelo matemático é uma função que é ajustada ao banco de dados visando descrever seus padrões. Podendo ser uma reta, uma função polinomial de um determinado grau, uma função exponencial, dentre várias outras opções.

5. **Função para avaliação do erro**
   Esta função é a responsável pelo ajuste do modelo ao conjuntos de dados de treinamento. O trabalho realizado quando se aborda o *treinamento* do modelo matemático citado anteriormente consiste essencialmente no processo de minimação desta função.

6. **Métrica de avaliação do treinamento**
   Após o treinamento do modelo para seu ajuste, fazemos então a aplicação do modelo no conjunto de teste para saber se o modelo é adequado.

Neste trabalho essencialmente serão abordados dois tipos de problemas de Aprendizagem de Máquina: Problemas de Classificação e Problemas de Regressão.

Os Problemas de Regressão consistem na previsão de algum valor contínuo, por exemplo, dado uma conjunto de valores e medidas sobre determinadas casas como a área, o número de quartos, número de banheiros, pode-se prever quanto pode custar o aluguel desta casa.

Outro exemplo também pode ser a previsão de quantos litros de combustível que determinado veículo gastará em uma certa distância tendo em vista algumas das condições do tal veículo.

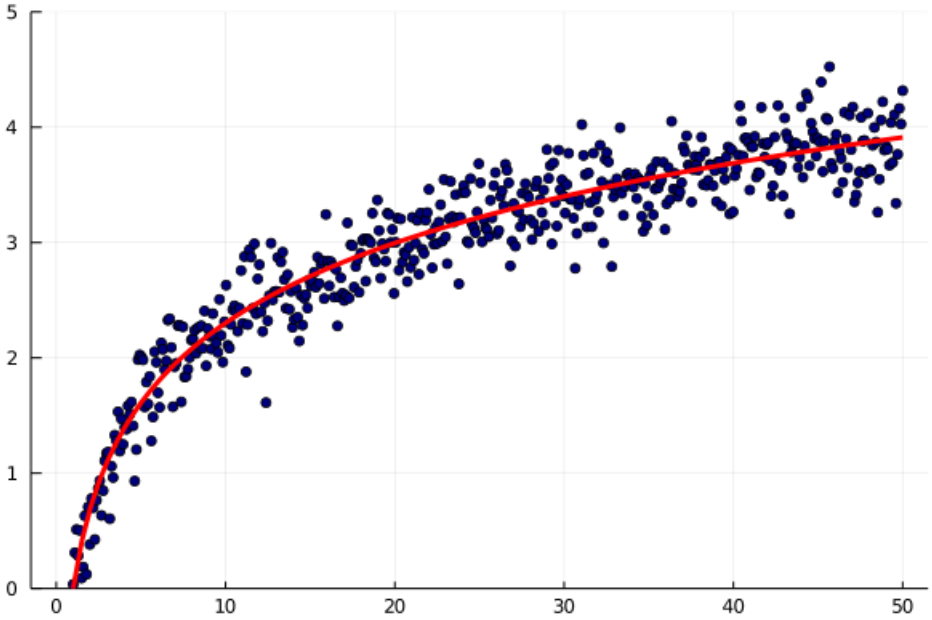


Figura 3: Exemplo de um problema de Regressão.

Nos Problemas de Classificação, o foco está em categorizar um conjunto de dados em duas ou mais classes diferentes, ou seja, como exemplos para este tipo de problema, podemos considerar que, dado o histórico de pagamentos e atrasos pelos clientes de um determinado banco, classificar novos clientes como possíveis bons ou maus pagadores; dadas algumas características morfológicas de células cancerígenas, classificá-las como possíveis cânceres benignos ou malignos.

Na classificação também se enquadram alguns problemas que envolvem visão computacional, ou seja, o computador aprendendo a interpretar imagens tais como o reconhecimento de caracteres matemáticos escritos a mão, a identificação de quais peças de roupas são camisetas ou calças, dentre outras possibilidades.

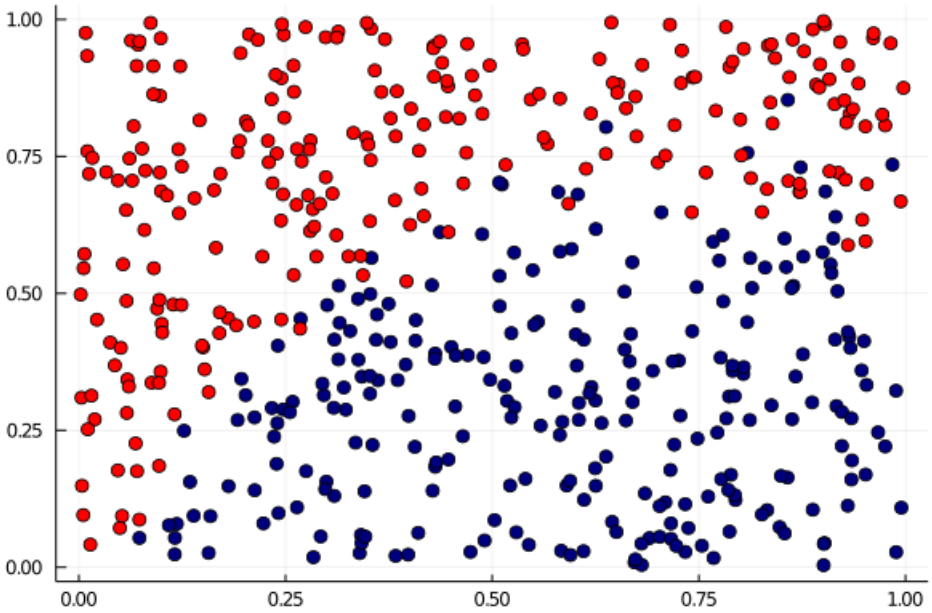


Figura 4: Exemplo de problema de Classificação.

Outro ponto importante para saber sobre a Aprendizagem computacional são os chamados *overfitting* e *underfitting*. Estes são problemas comuns enfrentados no procedimento de treinamento e teste do modelo matemático ajustado:

Overfitting: Este problema está associado a quando o modelo se ajusta excessivamente ao conjunto de dados de treinamento. A grosso modo, o modelo fica "viciado" nos dados de treinamento e obtém-se uma má pontuação de acertos no conjunto de teste.

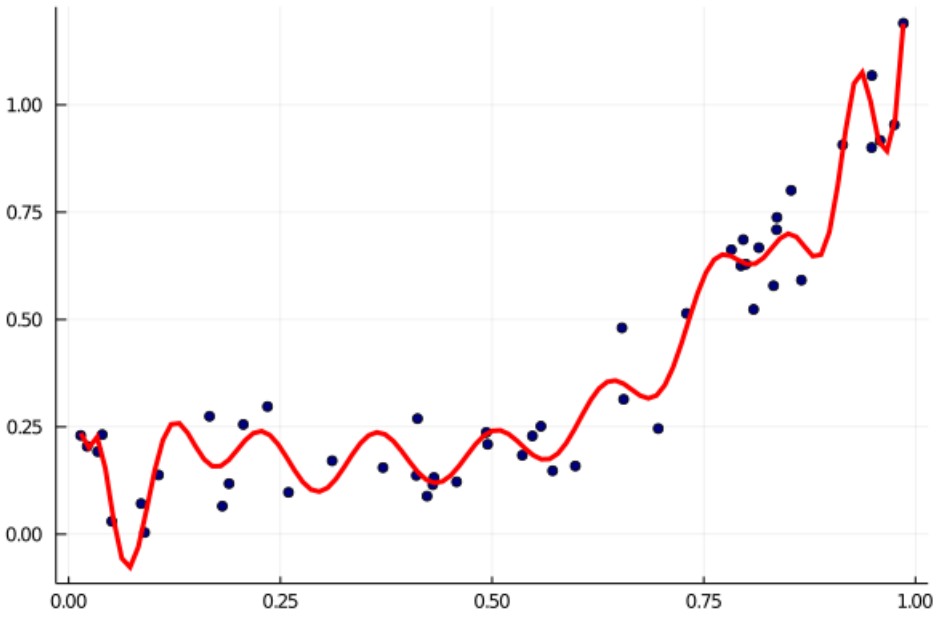


Figura 5: Exemplo de *Overfitting*.

Underfitting: Ao contrário do *Overfitting*, neste caso o modelo não se ajusta o suficiente aos dados de treino e consequentemente mantém uma taxa de acerto ruim no conjunto de dados de teste.

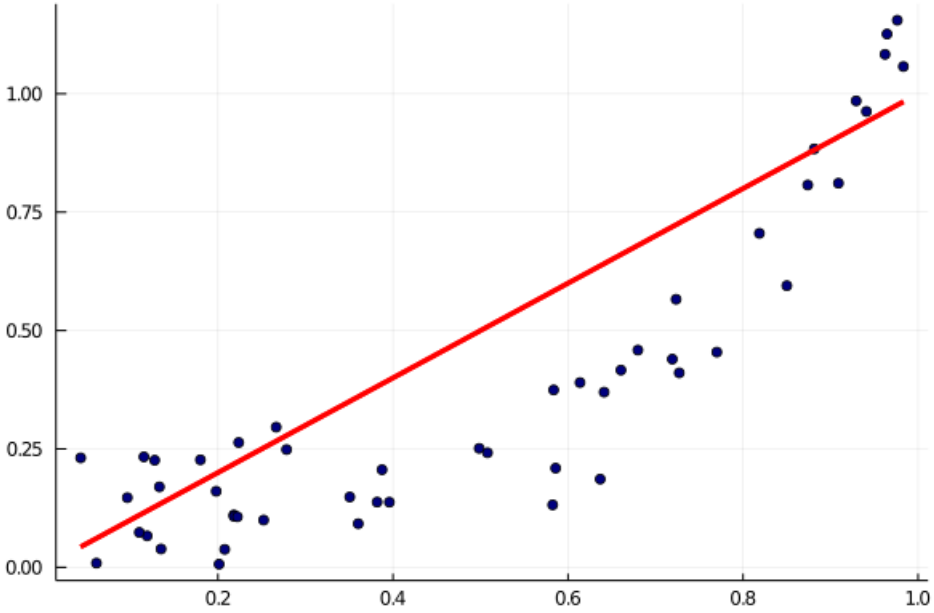


Figura 6: Exemplo de *Underfitting*.

Ou seja, o mais adequado sempre é encontrar um modelo que não esteja muito ajustado aos dados de treinamento (*Overfitting*), e também que não esteja distante de reproduzir os padrões do banco de dados (*Underfitting*).

2.1 Regressão e Quadrados Mínimos

Inicialmente, podemos pensar em um problema mais simples como determinar o modelo matemático que, quando ajustado, melhor descreve a relação entre os pontos representados no diagrama de dispersão da Figura 2.1.

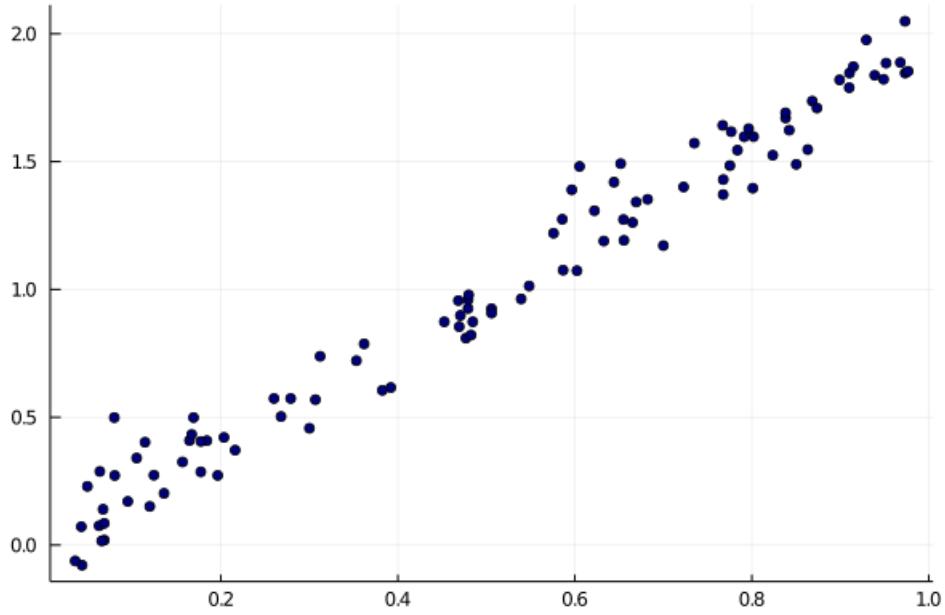


Figura 7: Diagrama de dispersão.

É possível notar que os pontos descrevem algo semelhante a uma reta. Sendo assim, o modelo mais plausível de se ajustar aos dados será da forma

$$\hat{y} = \beta_0 + \beta_1 x, \tag{1}$$

ao que nomeamos de *Regressão Linear*.

Para o treinamento do modelo, ou seja, o ajuste dos coeficientes β_0, β_1 é necessário algum método de minimização do erro entre os valores previstos pelo modelo e os valores reais a serem modelados, para isso, é aplicado o método conhecido como Método dos Quadrados Mínimos

- **Método dos Quadrados Mínimos Linear**
  Tendo em vista o problema então abordado, o ajuste dos coeficientes é feito a partir do erro quadrático. Assim, considere $y^{(i)}$ o valor original associado ao $i-$ésimo dado do conjunto de dados e $\hat{y}^{(i)}$ o valor estimado pelo modelo sobre o $i-$ésimo dado. O objetivo é encontrar $\hat{y}$ de modo que se obtenha o mínimo de

$$\sqrt{\frac{1}{2}\sum_{i=1}^{n}[y^{(i)} - \hat{y}^{(i)}]^2}.$$

  Se tratando de um modelo linear, consideramos

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

em que $\hat{\beta} = \text{argmin} \quad L(\beta)$ com

$$L(\beta) = \frac{1}{2}\sum_{i=1}^{n}\left[y^{(i)} - \hat{y}^{(i)}\right]^2 = \frac{1}{2}\sum_{i=1}^{n}\left[y^{(i)} - (\beta_0 + \beta_1 x^{(i)})\right]^2.$$

Encontrando este $\hat{\beta}$, teremos os coeficientes da reta que melhor descreve o conjunto de dados, como pode ser visualizado na Figura 8.

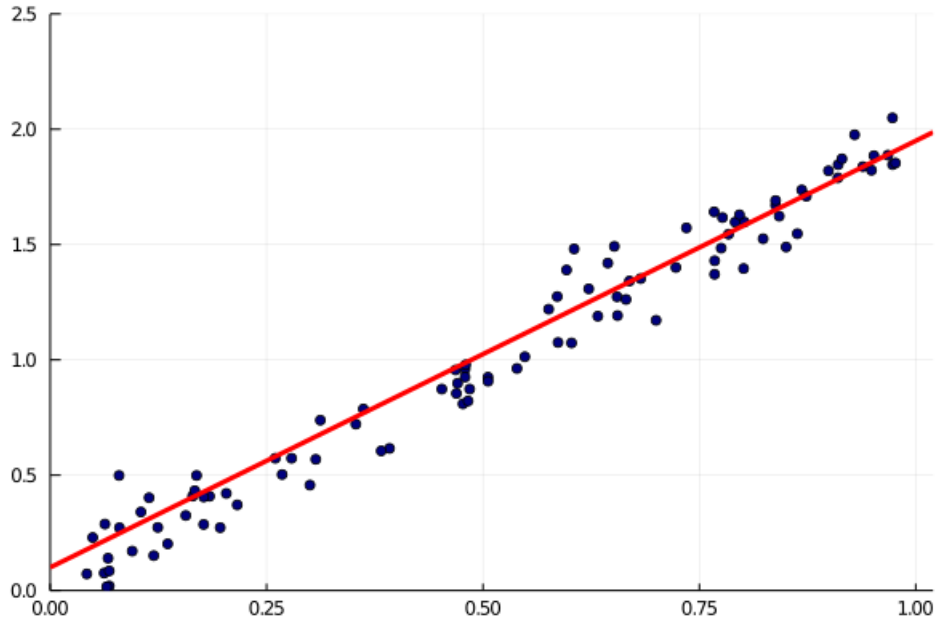


Figura 8: Ajuste da reta aos pontos.

Pode-se também generalizar o modelo e estender então para um caso Multilinear, ou seja, considere o conjunto de dados com n pares $(x^{(i)}, y^{(i)}) \in \mathbb{R}^p \times \mathbb{R}$ com $i = 1, \ldots, p$. Definimos a função $h_\beta(x)$ responsável pelas avaliações dos dados, ou seja, $h_\beta(x) = \hat{y}$. Esta função, neste caso, é dada por

$$h_\beta(x^{(i)}) = \beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \cdots + \beta_p x_p^{(i)}, \qquad \forall i = 1, \ldots, n.$$

O problema de quadrados mínimos se resume à solução de

$$L(\beta_0, \beta_1, \beta_2, \ldots, \beta_p) = \frac{1}{2}\sum_{i=1}^{n}[y^{(i)} - h_\beta(x^{(i)})]^2 \tag{2}$$

em que podemos escrever a diferença

$$y^{(i)} - h_\beta(x^{(i)}), \qquad \forall i = 1, \ldots, n$$

de forma matricial para

$$\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix} - \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_p^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_p^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \cdots & x_p^{(n)} \end{bmatrix} \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}, \tag{3}$$

e tomamos, respectivamente, como as matrizes y, X e β. Então, unindo (2) com (3), podemos escrever o problema como

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{2} \parallel y - X\beta \parallel_2^2$$

Porém, nem sempre é possível que a função $h_\beta(x)$ citada seja linear. Um exemplo simples deste fato é o próprio diagrama de dispersão dos casos de Covid-19 no Brasil em 2020,

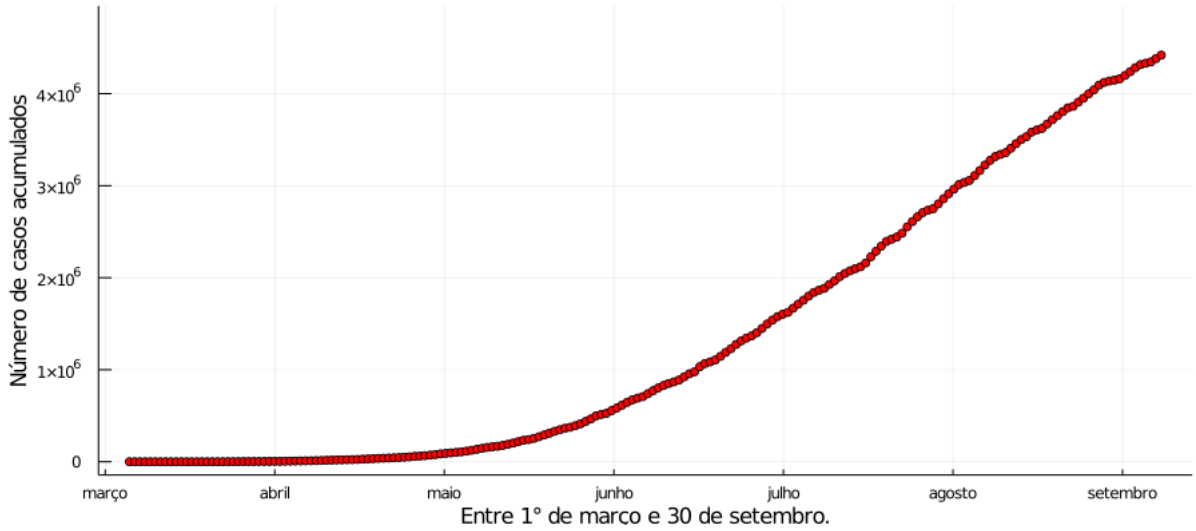


Figura 9: Número de casos de Covid-19 no Brasil entre os meses de março e setembro. Fonte: `https://covid.saude.gov.br` [4]

sendo assim, é necessário estender os casos anteriores. Podemos obter então o seguinte problema:

Seja o conjunto de dados com n pontos $(x^{(i)}, y^{(i)}) \in \mathbb{R}^p \times \mathbb{R}$ com $i = 1, \ldots, n$ e a função $h_\beta(x)$ não-linear em x. Escrevemos uma forma matricial

$$r(\beta) = \begin{bmatrix} y^{(1)} - h_\beta(x^{(1)}) \\ y^{(2)} - h_\beta(x^{(2)}) \\ \vdots \\ y^{(n)} - h_\beta(x^{(n)}) \end{bmatrix} \qquad \Longrightarrow \qquad L(\beta) = \frac{1}{2} \parallel r(\beta) \parallel_2^2,$$

e com isso, podemos colocar o problema como

$$\min_{\beta \in \mathbb{R}^p} L(\beta).$$

2.2 Classificação com Regressão Logística

Regressão logística consiste em resolver um problema de classificação binária, ou seja, é um método para resolução de problemas que consistem em apenas duas categorizações possíveis. Alguns exemplos de problemas abordados são: a liberação de um empréstimo bancário, classificada como *sim* ou *não*; A separação de células cancerígenas em *benignas* ou *malignas*; O acesso ou não a um determinado anúncio publicitário em determinados sites na internet. Generalizando, consideramos um evento $Y = 1$ para um dos casos e $Y = 0$ para o caso contrário.

Tomando então um evento binário com as possibilidades *sim* ou *não*, sabendo determinados dados x que implicam na probabilidade deste ocorrer, podemos escrever a probabilidade de *sim* dado x como

$$P(Y = \text{sim}|x),$$

podendo ser abreviado como $p(x)$ que consiste em um valor avaliado entre 0 e 1.

Agora, para identificar a função $p(x)$ que pode se ajustar melhor ao conjunto de dados binários, podemos avaliar o diagrama de dispersão na Figura 10 de um exemplo destes dados binários.

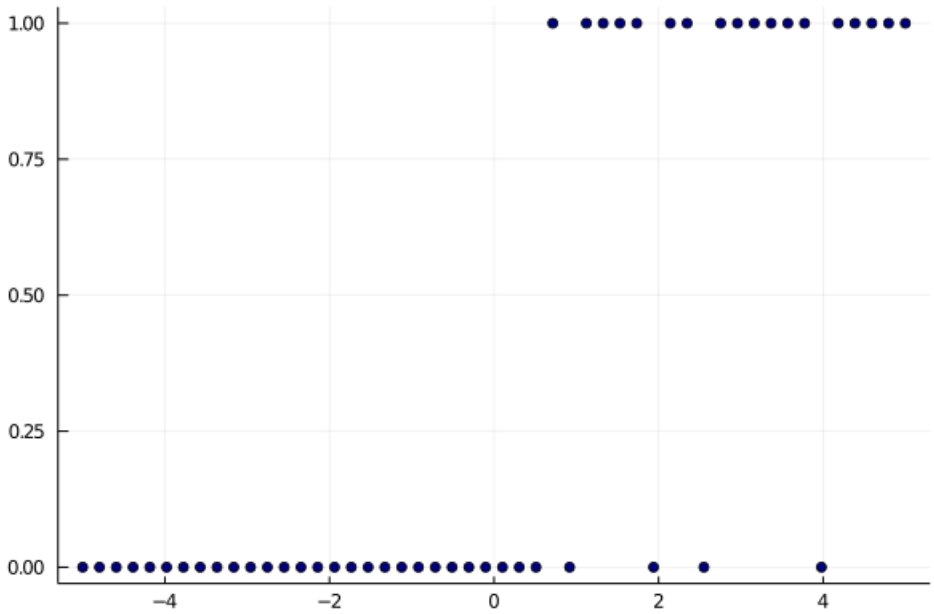


Figura 10: Exemplo de diagrama de dispersão de rótulos binários.

Note que uma reta como em um problema de Regressão Linear não descreveria com

tanta precisão este conjunto de dados, além de que poderia apresentar valores $p(x) > 1$ ou $p(x) < 0$. Para desviar destes problemas é necessário a introdução da chamada função logística, ou função *sigmoide* dada por

$$p(x) = \sigma(x) = \frac{1}{1 + e^{-x}},$$

que tem seu gráfico dado a seguir na Figura 11.

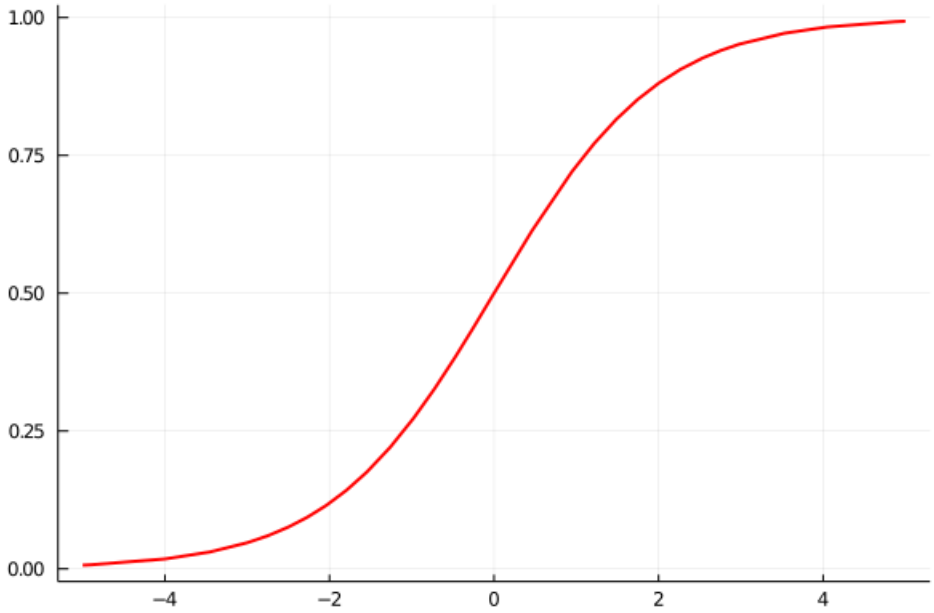


Figura 11: Gráfico da função sigmoide.

Porém, para poder ajustar o gráfico da melhor forma a conjuntos de pontos como o exemplo da Figura 10, é necessário acrescentar os coeficiente β_0, β_1, o que nos permite definir o modelo

$$h_\beta(x) = \sigma(\beta_0 + \beta_1 x) = \frac{1}{1 + e^{-\beta_0 - \beta_1 x}}.$$

O acréscimo destes coeficientes permite moldar melhor a função, visto que fixando β_1 e alterando β_0, tem-se o deslocamento da função pelo eixo x, como ilustrado na Figura 12.

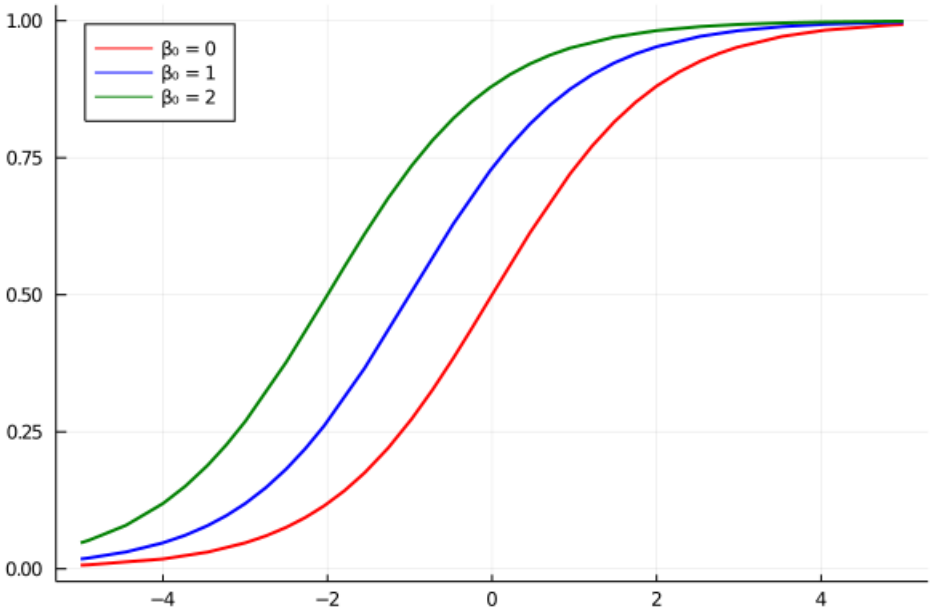


Figura 12: Variação do coeficiente β_0.

Fixando $\beta_0 = 0$ e variando β_1 temos que este é o responsável pela curvatura da função, como visto na Figura 13.

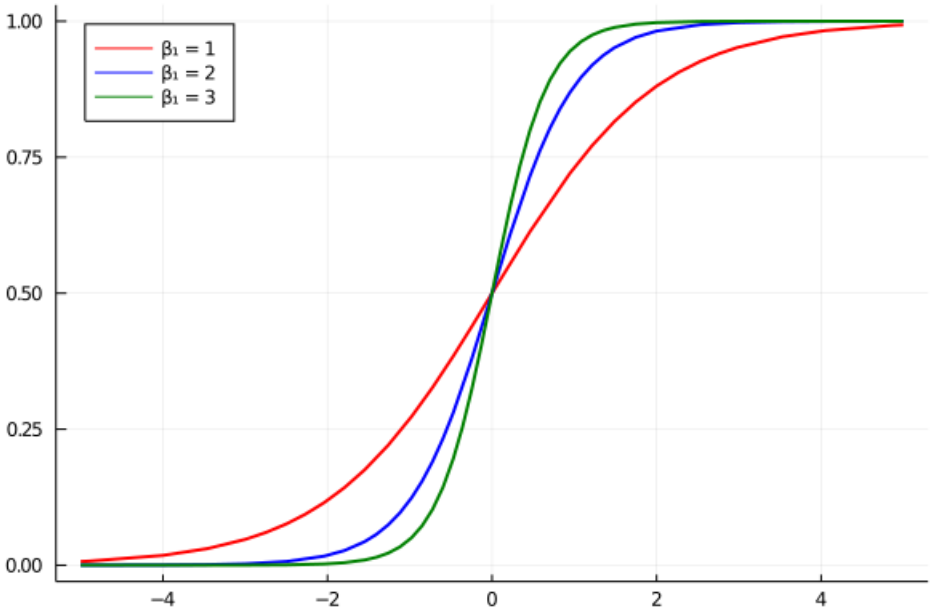


Figura 13: Variação do coeficiente β_1.

Agora, o problema consiste em estimar corretamente quais devem ser os valores em β_0, β_1 que melhor descrevem o conjunto de dados.

No problema de Regressão Linear, nesta etapa aplicava-se o Método dos Quadrados Mínimos Linear para a avaliação. Tendo em vista a não-linearidade da função sigmoide,

é possível aplicar o Método dos Quadrados Mínimos Não-Linear, ou seja, considerando $(x^{(i)}, y^{(i)}) \in \mathbb{R} \times \{0, 1\}$, tomamos a função a ser minimizada como

$$r(\beta_0, \beta_1) = \begin{bmatrix} y^{(1)} - h_\beta(x^{(1)}) \\ y^{(2)} - h_\beta(x^{(2)}) \\ \vdots \\ y^{(n)} - h_\beta(x^{(n)}) \end{bmatrix} \quad \Longrightarrow \quad L(\beta_0, \beta_1) = \frac{1}{2} \parallel r(\beta_0, \beta_1) \parallel^2,$$

em que podemos escrever o problema como

$$\min_{\beta_0, \beta_1 \in \mathbb{R}} L(\beta_0, \beta_1).$$

Porém, como é visto em (JAMES, 2013, p. 135) [7], devido a determinadas propriedades estatísticas, o método geralmente aplicado para o problema de Regressão Logística é o chamado de Método da Máxima Verossimilhança.

Este método consiste em encontrar os valores β_0 e β_1 de modo que a probabilidade $h_\beta(x^{(i)})$ prevista para dado i seja a mais próxima possível da informação já observada deste dado. Por exemplo, para dados i que tenham como observado o rótulo *sim* ($y^{(i)} = 1$), o valor de $h_\beta(x^{(i)})$ é mais próximo de 1, já para dados i que recebiam o rótulo *não* ($y^{(i)} = 0$), $h_\beta(x^{(i)})$ deve avaliar valores próximos a 0.

Formalizando matematicamente este método, definimos a função de Verossimilhança

$$\ell(\beta_0, \beta_1) = \prod_{i:y^{(i)}=1} h_\beta(x^{(i)}) \prod_{i:y^{(i)}=0} \left(1 - h_\beta(x^{(i)})\right)$$

então, os valores de β_0, β_1 são estimados de modo a maximizar esta função.

Ainda, como pode ser visto em (MEYER, 1987, p. 339) [10], considerando $i = 1, \ldots, n$, podemos escrever esta equação como

$$\ell(\beta) = \prod_{i=1}^{n} h_\beta(x^{(i)})^{y^{(i)}} \left(1 - h_\beta(x^{(i)})\right)^{1-y^{(i)}},$$

e então, para facilitar o calculo do valor de máximo, tomamos

$$L(\beta) = \log \ell(\beta),$$

que, com a aplicação de algumas propriedades logarítmicas, finalmente obtemos a função

$$L(\beta) = \sum_{i=1}^{n} \left[y^{(i)} \log\left(h_\beta(x^{(i)}\right) + (1 - y^{(i)}) \log\left(1 - h_\beta(x^{(i)})\right) \right]. \tag{4}$$

Nesta função, note que, para algum dado i, se $y^{(i)} = 0$, teremos $h_\beta(x^{(i)})$ tendendo para 0 o que implica em $\log\left(1 - h_\beta(x^{(i)})\right)$ tende a 0. E também, tendo um dado i tal que $y^{(i)} = 1$, então enquanto $h_\beta(x^{(i)})$ tende para 1, $\log\left(h_\beta(x^{(i)})\right)$ tende a 0.

Com os valores em β encontrados, recebendo um dado $i+1$, será possível determinar se este receberá um rótulo de *sim* ou *não* a partir dos valores de $x^{(i+1)}$.

Agora, com todo este processo apresentado, é possível generalizar o problema, ou seja, considere uma matriz $X \in \mathbb{R}^{n\times p+1}$, o vetor de coeficientes $\beta = (\beta_0, \beta_1, \ldots, \beta_p)$. Teremos que os valores avaliados pelo modelo são dados por

$$\hat{y}^{(i)} = h_\beta(x^{(i)}) = \sigma({x^{(i)}}^T\beta), \quad \forall i = 1, \ldots, n.$$

Ou utilizando um abuso de notação podemos escrever a função aplicada em todas as linhas da matriz X como

$$\hat{y} = h_\beta(X) = \sigma(X\beta).$$

Finalmente, utilizando a equação (4) e resolvendo então o problema de otimização dado por

$$\max_{\beta\in\mathbb{R}^p} L(\beta),$$

obtemos os valores mais adequados para o vetor β e com estes valores é possível realizar as classificações de novos dados após atualizarmos β em nosso modelo $h_\beta(x^{(i)})$.

Sobre a rotulação dos resultados, como a função $h_\beta(x^{(i)})$ apenas entrega um valor probabilístico entre 0 e 1, é necessário atribuir corretamente os rótulos 1 e 0 para tais valores. Para isso, é estabelecido um limiar η, que comumente recebe o valor de 0,5. Com isso, é possível determinar uma função $c(x^{(i)})$ de classificação dada por

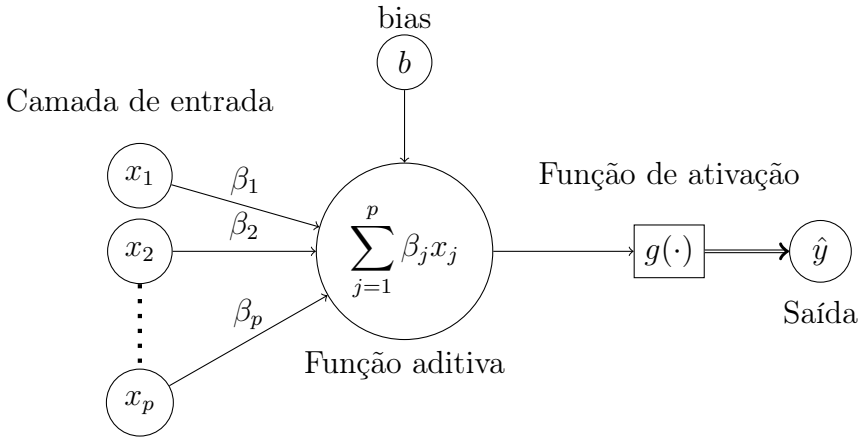
$$c(x) = \begin{cases} 1, & \text{se} \quad \sigma(x^T\beta) \geq \eta \\ 0, & \text{caso contrário,} \end{cases} \qquad \forall x \in \mathbb{R}^p.$$

Após este processo de rotulação das probabilidades, a resolução do problema de Regressão Logística está completa.

2.3 Redes Neurais - O Perceptron Multicamadas

Outro modelo matemático muito utilizado na Aprendizagem de Máquina é o chamado de Redes Neurais Artificiais. Este modelo, como seu próprio nome sugere, tem sua inspiração na em redes neurais reais do cérebro humano.

Um neurônio humano é composto essencialmente por um dendrito, um corpo celular e o axônio. A informação é passada pelo neurônio através de sinapses: o dendrito recebe o impulso sináptico de outros neurônios, repassa ao corpo celular e este o envia ao axônio que transmite então para outro neurônio repetindo o processo. Para simular este neurônio real, em uma rede neural artificial utilizamos o chamado *perceptron*:



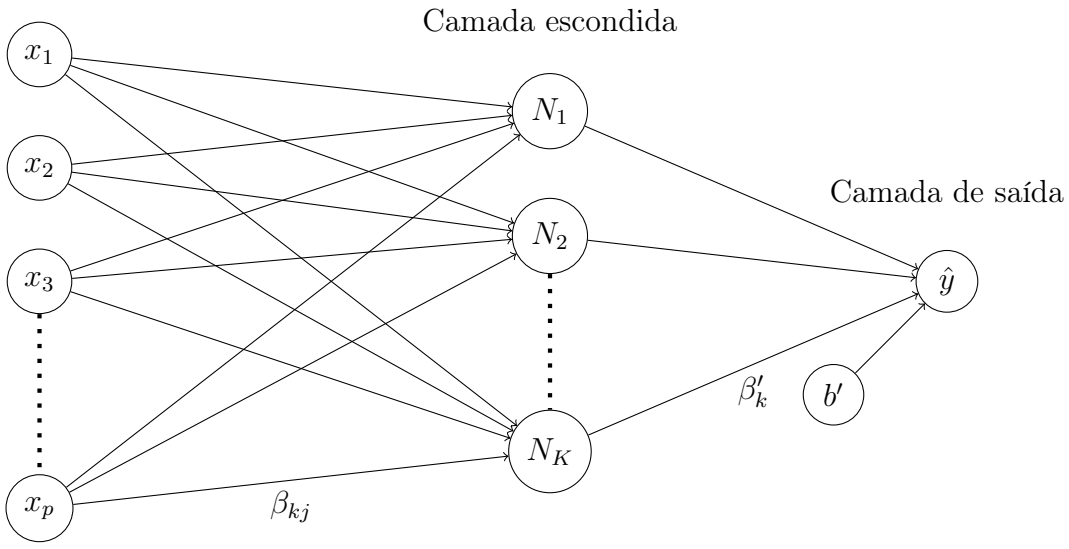
Temos a camada de entrada onde entram os dados iniciais x_j, com $j = 1, \ldots, p$ seguida então dos pesos sinápticos β_j e da função aditiva, que efetua a soma dos produtos dos pesos sinápticos β_j com os dados x_j. Seguida desta adição é aplicada uma função de ativação que determina o valor de saída do neurônio.

Matematicamente, transcrevemos este neurônio artificial como uma função $N : \mathbb{R}^p \to \mathbb{R}$ dada por

$$N(x) = g\left(\sum_{j=1}^{p} \beta_j x_j + b\right) \tag{5}$$

em que $g : \mathbb{R} \to \mathbb{R}$ é a função de ativação citada anteriormente e b é o valor de *viés*.

Porém, tal como um cérebro humano, não utiliza-se apenas um nerônio deste, sendo necessária então a construção de uma rede de neurônios.



Novamente, podemos transcrever esta rede visualizada para linguagem matemática, obtendo-se

$$h_\beta(x) = g_2\left(\sum_{k=1}^{K} \beta'_k N_k(x) + b'\right) \tag{6}$$

em que,

$$N_k(x) = g_1\left(\sum_{j=1}^{p} \beta_{kj} x_j + b_k\right), \qquad k = 1, \ldots, K. \tag{7}$$

Cada função N_k representa o resultado de um neurônio dos K neurônios apresentados no exemplo, e g_1, g_2 são as funções de ativação.

É possível escrever esta estrutura da Rede Neural de forma mais generalizada, considerando então $x \in \mathbb{R}^p$, a matriz $B \in \mathbb{R}^{K\times p}$ com $(B)_{k,j} = \beta_{kj}$ e o vetor $b \in \mathbb{R}^K$, de modo que o vetor formado na camada escondida é dado por

$$g_1(Bx + b) \in \mathbb{R}^K,$$

seguido então da avaliação para a camada de saída. Neste caso, como a saída consiste em apenas um elemento, consideramos uma matriz linha $B' \in \mathbb{R}^{1\times K}$ e $b' \in \mathbb{R}$, escrevendo então

$$g_2(B' \cdot g_1(B \cdot x + b) + b') = \hat{y}.$$

Ou seja, considerando uma Rede Neural com duas camadas escondidas, por exemplo, teremos as matrizes $B^{(1)}, B^{(2)}, B^{(3)}$ dos pesos sinápticos das camadas e os valores b_1, b_2, b_3 de vieses também respectivamente de cada camada. Dado um vetor x então, teremos

$$g_3(B^{(3)} \cdot g_2(B^{(2)} \cdot g_1(B^{(1)} \cdot x + b_1) + b_2) + b_3) = \hat{y},$$

com g_1, g_2, g_3 funções de ativação.

Finalmente, generalizando para uma Rede Neural Multicamada com m camadas escondidas, teremos

$$g_{m+1}(B^{(m+1)} \cdot g_m(B^{(m)} \cdots g_1(B^{(1)}x + b_1) \cdots + b_m) + b_{m+1}) = \hat{y},$$

sendo g_i as funções de ativação.

A implementação de algoritmos de Redes Neurais Multicamadas não é uma tarefa fácil. As diversas funções compostas que são necessárias para o funcionamento da rede fazem com que, caso sejam mal implementadas, tragam muitos custos computacionais e um processo de otimização demasiadamente lento, não sendo então tão efetivo. Sendo assim, tendo em vista que o foco deste trabalho é a implementação do Método do Gradiente Estocástico, para não fugir muito ao tema, foi modelada e trabalhada uma Rede Neural mais simples com uma camada escondida de K neurônios. Com esta simples estrutura já

é possível fazer uma aplicação e resolver de maneira bastante satisfatória problemas de Regressão Linear e Regressão Logística.

Agora, quanto à escolha das funções de ativação $g_i : \mathbb{R} \to \mathbb{R}$, há diversas opções diferentes, sendo a escolha é feita de acordo com o problema a ser resolvido. Algumas opções aqui trabalhadas, além da própria *sigmoide*, são as funções:

- **ReLU (função da Unidade Linear Retificada):**

$$g_i(x) = \max\{0, x\}$$

  Consiste essencialmente em deixar de lado os números negativos. Tendo em vista um problema de classificação onde o valor esperado de g_i tem interpretação probabilística, esta função pode ser bastante interessante.

- **Tangente Hiperbólica:**

$$g_i(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

  Esta função tem uma aplicação muito semelhante á função *sigmoide*, porém com o a imagem no intervalo $[-1, 1]$.

- **Função identidade:**

$$g_i(x) = x$$

  Por mais simples que seja, essa função pode ser aplicada no contexto dos problemas de Regressão Linear por exemplo, pois os valores obtidos da Rede Neural não devem ser classificados ou categorizados de alguma forma, isto é, os valores consistem de valores contínuos.

2.4 Padronização dos problemas

Agora, tendo uma visão mais geral dos problema apresentados, é possível notar que todos seguem um mesmo padrão de abordagem, com a utilização de $h_\beta(x)$ que define a função matemática mais adequada para ser ajustada ao conjunto de dados com a atualização dos coeficientes em β, e de uma função $L(\beta)$ que faz a comparação dos valores avaliados pelo modelo com os valores reais do conjunto de dados de treino, ou seja, avalia o erro do modelo.

Assim, pode-se generalizar as notações utilizadas definindo os problemas como

1. **Um modelo**
   Uma função $h_\beta(x)$ escolhida de forma que mais se adeque a cada tipo de problema a ser resolvido (classificação ou regressão), e depende dos parâmetros β que são atualizados pelo algoritmo de acordo com aumento das iterações.

2. **Avaliação do erro**
   Função definida por $\ell(y, \hat{y})$, que faz a avaliação do erro de uma única avaliação, ou seja, a função calcula o erro entre o valor original $y \in \mathbb{R}$ do conjunto de dados e o valor avaliado pelo modelo proposto $\hat{y} = h_\beta(x)$.

3. **Função de perda**
   Esta consiste na avaliação do erro médio do modelo. Tendo em vista que a função $\ell(y, \hat{y})$ citada avalia o erro sob uma única previsão, considera-se então a função dada por

$$L(\beta) = \frac{1}{n} \sum_{i=1}^{n} \ell(y^{(i)}, h_\beta(x^{(i)})).$$

Agora, podemos escrever cada problema utilizando destas notações.

1. **Regressão Linear**
   Definimos

$$h_\beta(x) = x^T \beta,$$

   com

$$\ell(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2,$$

   que implica em

$$L(\beta) = \frac{1}{2} \parallel y - h_\beta(X) \parallel^2 .$$

2. **Regressão Logística**
   Neste caso temos

$$h_\beta(x) = \sigma(x^T \beta) = \frac{1}{1 + e^{-x^T \beta}},$$

   com erro avaliado por

$$\ell(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}),$$

   e a função perda

$$L(\beta) = -\frac{1}{n} \sum_{i=1}^{n} \left[y^{(i)} \log(h_\beta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\beta(x^{(i)})) \right].$$

3. **Rede Neural**
   A função é definida como

$$h_\beta(x) = g_{m+1}(B^{(m+1)} \cdot g_m(B^{(m)} \cdots g_1(B^{(1)}x + b_1) \cdots + b_m) + b_{m+1}),$$

   em que $g_i, \forall i = 1, \ldots, m+1$ são as funções de ativação escolhidas de acordo com o problema. Para este caso, as funções $\ell(y, \hat{y})$ e $L(\beta)$ utilizadas são as mesmas das já apresentadas anteriormente, sendo escolhidas de acordo com o problema, ou seja, classificação binária ou regressão.

Para uma maior praticidade da notação, pode-se definir

$$L_i(\beta) = \ell(y^{(i)}, h_\beta(x^{(i)})),$$
$$\Rightarrow L(\beta) = \frac{1}{n}\sum_{i=1}^{n} L_i(\beta).$$

Esta notação facilitará a compreensão dos algoritmos futuramente no trabalho.

3 Otimização

A Otimização é um ramo da Matemática que consiste no estudo de problemas dos quais suas soluções se baseiam na maximização ou minimização de uma determinada função, conhecida como função objetivo. Estes problemas podem ser separados em restritos e irrestritos.

Os problemas do tipo restritos consistem na maximização ou minimização da função objetivo considerando algumas restrições para o domínio da mesma, por exemplo, podemos considerar

$$\begin{aligned} \min_{x \in \mathbb{R}^2} \quad & f(x) = x_1 + x_2 \\ \text{sujeito a} \quad & x_1^2 + x_2^2 \leq 2. \end{aligned}$$

Porém, no contexto dos problemas de Aprendizagem de Máquina a serem trabalhados neste trabalho, considera-se otimização irrestrita, ou seja, considerando uma função objetivo $f : \mathbb{R}^n \to \mathbb{R}$, podemos escrever tais problemas como

$$\min_{x \in \mathbb{R}^n} f(x).$$

Com isso, visando então a busca deste valor que minimiza a função f, parte-se da seguinte proposição (FRIEDLANDER, 1994, p. 11)[6]

Proposição 3.1. *Seja* $f : \mathbb{R}^n \to \mathbb{R}, f \in C^1$, *Se* x^* *é um minimizador local de* f *em* $\mathbb{R}^n$ *então* $\nabla f(x^*) = 0$.

Em que, também em (FRIEDLANDER, 1994, p. 7)[6], define-se minimizador local como

Definição 1. *Um ponto* $x^* \in S$ *é um minimizador local de* f *em* S *se e somente se existe* $\varepsilon > 0$ *tal que* $f(x) \geq f(x^*)$ *para todo* $x \in S$ *tal que* $\| x - x^* \| < \varepsilon$.

Porém, encontrar o valor que minimiza a função nem sempre é uma tarefa fácil, para auxiliar nesta busca são apresentados neste trabalho os métodos tradicionais no contexto da Aprendizagem de Máquina, estes são o Gradiente Descendente e Gradiente Estocástico.

3.1 Método do gradiente e sua convergência

Para compreender melhor a funcionalidade do método do Gradiente Descendente, é necessário entender como definimos as atualizações do coeficiente β. Sendo assim, considere inicialmente o problema da forma

$$\min_{\beta \in \mathbb{R}^p} f(\beta).$$

Para as hipóteses, lema e demonstração a seguir, são utilizadas como referências (ZHOU, 2018, p. 4) [17], (BOTTOU; NOCEDAL, 2018, p. 23, 81) [3] e no lema 1.2.3 de (NESTEROV, 2018, p. 25-26) [11].

Hipótese 1: A função f é diferenciável e seu gradiente, $\nabla f : \mathbb{R}^p \to \mathbb{R}^p$, é uma função *L-Lipschitz*, ou seja, existe $L > 0$ tal que

$$\| \nabla f(\hat{\beta}) - \nabla f(\beta) \| \quad \leq \quad L \| \hat{\beta} - \beta \| \qquad \forall \hat{\beta}, \beta \in \mathbb{R}^p. \tag{8}$$

Hipótese 2: A função f é limitada inferiormente por $f_{min} \in \mathbb{R}$, ou seja, $f(\beta) \geq f_{min}, \forall \beta \in \mathbb{R}^p$.

Com isso, temos a atualização dada a partir do seguinte lema.

Lema 3.1. *Se a* ***Hipótese 1*** *vale, então*

$$|f(\hat{\beta}) - f(\beta) - \nabla f(\beta)^T(\hat{\beta} - \beta)| \quad \leq \quad \frac{L}{2} \| \hat{\beta} - \beta \|_2^2, \quad \forall \hat{\beta}, \beta \in \mathbb{R}^p \tag{9}$$

Demonstração. Dados $\hat{\beta}, \beta \in \mathbb{R}^p$, pelo Teorema do Valor Médio

$$f(\hat{\beta}) = f(\beta) + \int_0^1 \nabla f(\beta + t(\hat{\beta} - \beta))^T(\hat{\beta} - \beta) dt.$$

Subtraindo $\nabla f(\beta)^T(\hat{\beta}-\beta)$ de ambos os lados da igualdade, teremos

$$f(\hat{\beta})-f(\beta)-\nabla f(\beta)^T(\hat{\beta}-\beta)=\int_0^1\Big(\nabla f(\beta+t(\hat{\beta}-\beta))-\nabla f(\beta)\Big)^T(\hat{\beta}-\beta)dt$$

utilizando então a **Hipótese 1** e a desigualdade de *Cauchy-Schwartz* temos

$$\begin{aligned}|f(\hat{\beta})-f(\beta)-\nabla f(\beta)^T(\hat{\beta}-\beta)| &\leq \parallel\hat{\beta}-\beta\parallel_2\int_0^1\parallel\nabla f(\beta+t(\hat{\beta}-\beta))-\nabla f(\beta)\parallel_2 dt\\ &\leq L\parallel\hat{\beta}-\beta\parallel_2^2\int_0^1 tdt\\ &= \frac{L}{2}\parallel\hat{\beta}-\beta\parallel_2^2.\end{aligned}$$

□

Pelo lema, teremos que, dado $\beta^{(k)}\in\mathbb{R}^p$

$$f(\hat{\beta})\leq f(\beta^{(k)})+\nabla f(\beta^{(k)})^T(\hat{\beta}-\beta^{(k)})+\frac{L}{2}\parallel\hat{\beta}-\beta^{(k)}\parallel_2^2,\quad\forall\hat{\beta}\in\mathbb{R}^p.$$

Portando, definindo

$$\beta^{(k+1)} = \arg\min_{\hat{\beta}\in\mathbb{R}^p}\left\{f(\beta^{(k)})+\nabla f(\beta^{(k)})^T(\hat{\beta}-\beta^{(k)})+\frac{L}{2}\parallel\hat{\beta}-\beta^{(k)}\parallel_2^2\right\}$$

então, com o lado direito da igualdade, tomando a derivada em relação a $\hat{\beta}$ e igualando a zero, obtemos

$$\beta^{(k+1)}=\beta^{(k)}-\frac{1}{L}\nabla f(\beta^{(k)}).$$

Teremos então $f(\beta^{(k+1)})>f(\beta^{(k)}$ se $\nabla f(\beta^{(k)})\neq 0$. Tendo em vista a dificuldade de determinar exatamente este valor de L, tomamos $\frac{1}{L}=\gamma$, um hiperparâmetro que é decidido de forma empírica. Com isso, este resultado motiva o método iterativo de nome Gradiente Descendente dado pelo seguinte algoritmo.

Algoritmo 1: Método do Gradiente Descendente

Dados: Escolha $\beta^{(0)} \in \mathbb{R}^p$ e $\gamma, \varepsilon \in \mathbb{R}_+^*$, defina $k = 0$;

Entrada: Calcule $\nabla f(\beta^{(k)})$ para $k = 0$;

enquanto $\| \nabla f(\beta^{(k)}) \|_2 > \varepsilon$ **faça**

  Calcule $d^{(k)} = \nabla f(\beta^{(k)})$

  $\beta^{(k+1)} \leftarrow \beta^{(k)} - \gamma \cdot d^{(k)}$

  $k \leftarrow k + 1$

fim

retorna $\beta^{(k)}$

O valor de γ tratado no algoritmo recebe o nome de taxa de aprendizado. Como é explicado em (PATTERSON; GIBSON, 2017, p. 79), um valor grande para este parâmetro pode diminuir o tempo inicialmente porém, pode ser problemático caso leve o trajeto a ultrapassar o ponto mínimo. Por outro lado, considerar taxas pequenas pode levar o algoritmo a atingir bons valores porém, pode levar muito tempo e cálculos excessivos.

Este valor, quando tomado suficientemente pequeno, nos dá a validação do teorema a seguir. Sua demonstração também pode ser vista em (NESTEROV, 2018, p. 29-31) [11].

Teorema 3.1. *Suponha que a* ***Hipótese 1*** *e a* ***Hipótese 2*** *são válidas, considere* $\{\beta^{(k)}\}$ *uma sequência gerada pelo algoritmo do Gradiente Descendente. Se* $0 < \gamma < \frac{L}{2}$*, então*

$$\lim_{k \to +\infty} \| \nabla f(\beta^{(k)}) \|_2 = 0.$$

Demonstração:

Demonstração. A partir da **Hipótese 1** e do Lema 1 temos que

$$\begin{aligned} f(\beta^{(k+1)}) &\leq f(\beta^{(k)}) + \nabla f(\beta^{(k)})^T(\beta^{(k+1)} - \beta^{(k)}) + \frac{L}{2} \| \beta^{(k+1)} - \beta^{(k)} \|_2^2 \\ &= f(\beta^{(k)}) - \gamma \nabla f(\beta^{(k)})^T \nabla f(\beta^{(k)}) + \frac{L\gamma^2}{2} \| \nabla f(\beta^{(k)}) \|_2^2 \\ &= f(\beta^{(k)}) - \gamma \left(1 - \frac{L\gamma}{2}\right) \| \nabla f(\beta^{(k)}) \|_2^2 \end{aligned}$$

tomando $\gamma \in (0, \frac{2}{L})$, temos

$$f(\beta^{(k)}) - f(\beta^{(k+1)}) \quad \geq \quad \gamma \left(1 - \frac{L\gamma}{2}\right) \| \nabla f(\beta^{(k)}) \|_2^2 \quad \geq \quad 0$$

Com isso, temos que $\{f(\beta^{(k)})\}_{k \geq 0}$ é uma sequência monótona não-crescente e, tendo em vista a **Hipótese 2**, limitada inferiormente. Portanto, a sequência é convergente.

Podemos então tomar

$$\lim_{k \to +\infty} f(\beta^{(k)}) = f^*$$

que teremos

$$\lim_{k \to +\infty} f(\beta^{(k)}) - f(\beta^{(k+1)}) = f^* - f^* = 0$$

e então, tendo em vista $f(\beta^{(k)}) - f(\beta^{(k+1)}) \geq 0$ e sua convergência a 0, temos pelo Teorema do confronto que

$$\lim_{k \to +\infty} \| \nabla f(\beta^{(k)}) \|_2 = 0.$$

□

Portanto, tem-se a garantia de que para um k suficientemente grande, ou seja, uma número de iterações grande o suficiente, o Algoritmo 1 encontra um minimizador aproximado.

Uma forma mais intuitiva de se pensar sobre a funcionalidade do algoritmo pode ser dada partindo das curvas de nível a seguir nas Figuras 14 e 15

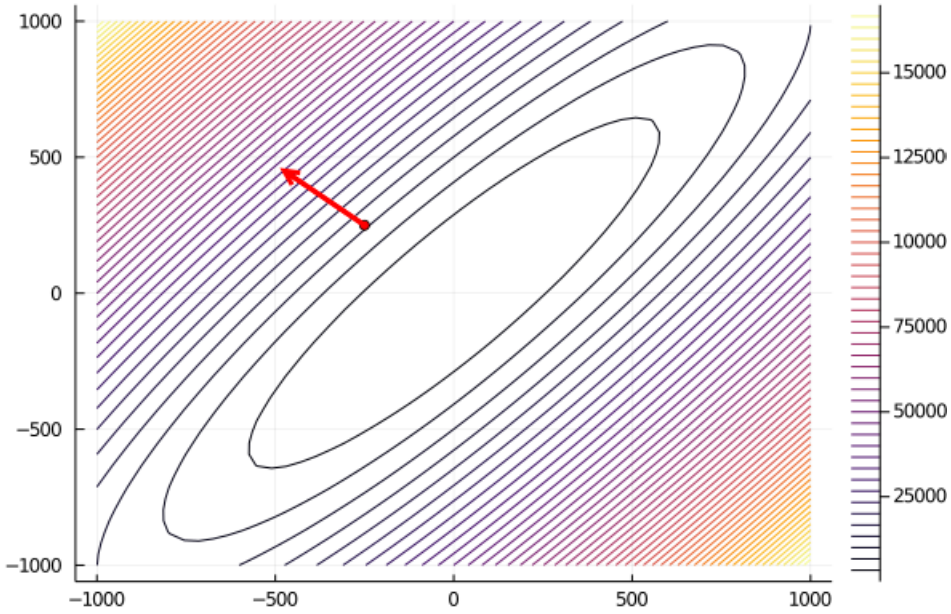


Figura 14: Gradiente com o próprio sinal.

Note que o gradiente de uma função f aponta para o "lado de fora" das curvas, sendo assim, quando tomamos a direção $-\nabla f(\beta)$ estamos apontado para o "dentro" das curvas, para onde pretendemos prosseguir.

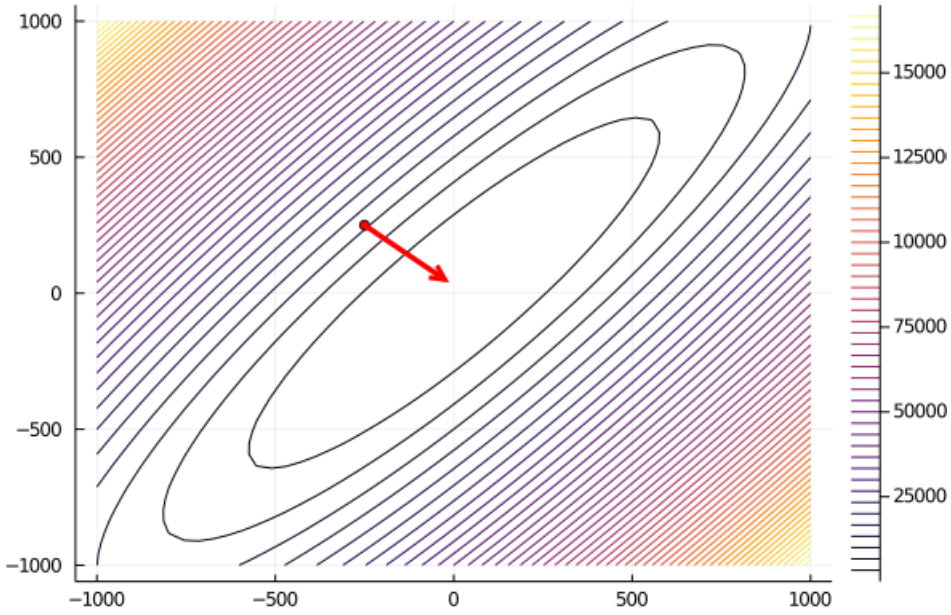


Figura 15: Gradiente com sinal oposto.

Além disso, note que o hiperparâmetro γ indica o tamanho do passo que caminhamos em direção ao ponto de minimização, o que evita que possam ocorrer passos exageradamente grandes e que passem direto pelo ponto de mínimo Além disso, o passo pode ter o tamanho alterado entre uma iteração e outra, sendo assim definimos aqui γ_k como o valor da taxa de aprendizado na iteração k.

3.2 O problema do Método do Gradiente Descendente

Apesar do método conseguir atingir o objetivo de minimizar a função retornando um valor ótimo para β, note que a atualização dos coeficientes ocorre da seguinte forma:

$$\beta^{(k+1)} = \beta^{(k)} - \gamma_k \cdot \nabla L(\beta^{(k)})$$

em que L é a função que está calculando o erro do valor avaliado com relação ao valor original do conjunto de dados, ou seja, para cada atualização do vetor β é avaliado o erro do modelo no conjunto total de dados e ainda, calculado o vetor gradiente sobre todo o conjunto de dados.

Quando levamos em consideração bancos de dados muito grandes, este processo pode se tornar muito custoso tanto no sentido computacional quanto no sentido tempo. Com isso, acaba não sendo o método mais indicado para a minimização da função.

3.3 Gradiente estocástico e sua convergência

Tendo em vista o problema do Gradiente Descendente, o Método do Gradiente Estocástico [8] visa atualizar o vetor β de acordo com o erro de cada previsão, ou seja, com o valor estimado sobre um dado x, atualiza-se o vetor β a partir da minimização do erro apenas para este dado. Colocando matematicamente, temos o seguinte processo:

Dada a matriz $X \in \mathbb{R}^{n\times p}$, o vetor $\beta \in \mathbb{R}^p$, o vetor de rótulos $y \in \mathbb{R}^n$, o modelo $h_\beta(x)$ escolhido e a função que avalia o erro $\ell(y, h_\beta(x))$, fazemos a atualização

$$\beta^{(k+1)} = \beta^{(k)} - \gamma \cdot \nabla L_i(\beta), \qquad \forall i = 1, \ldots, n.$$

Para a implementação do algoritmo, o dado $x^{(i)}$ é escolhido aleatoriamente para que o modelo não receba mais influencia dos valores iniciais de x. Sendo assim, o algoritmo fica da forma

Algoritmo 2: Método do Gradiente Estocástico

Dados: Escolha $\beta^{(0)} \in \mathbb{R}^p, \gamma_k \in \mathbb{R}^*_+$;
para $k = 1, \ldots, K$ **faça**
  S = *shuffle(1:n)*
  para $i \in S$ **faça**
    Calcule $d^{(k)} = -\nabla L_i(\beta)$;
    $\beta^{(k+1)} \leftarrow \beta^{(k)} + \gamma_k \cdot d^{(k)}$;
  fim
fim
retorna $\beta^{(K)}$

Para entender a convergência do Gradiente Estocástico, são necessárias algumas definições e lemas:

Definição 2. *Uma variável aleatória X é qualquer função definida sobre um espaço amostral Ω que atribui valor real a cada elemento do espaço amostral.*

Definição 3. *Uma variável aleatória X é definida como sendo discreta quando o número de valores possível que a variável assume for finito ou infinito enumerável.*

Definição 4. *A função de probabilidade de uma variável aleatória discreta é uma função que atribui probabilidade a cada um dos possíveis valores assumidos pelas variáveis. Isto é, sendo X uma variável aleatória discreta com valores $x_1, x_2, \ldots$, temos para $i \geq 1$,*

$$p(x_i) = P(X = x_i) = P(\{\omega \in \Omega | X(\omega) = x_i\}),$$

em que $p(x_i)$ satisfaz

1. $0 \leq p(x_i) \leq 1, \forall i \geq 1$;

2. $\sum_i p(x_i) = 1$.

Definição 5. *Seja X uma variável aleatória discreta com valores possíveis $x_1, x_2, \ldots$. Sejam $p(x_i) = P(X = x_i), \forall i \geq 1$. Então o valor esperado de X (também chamado esperança matemática de X ou média de X) é dado por*

$$\mathbb{E}_X(X) = \sum_{i=1}^{\infty} x_i p(x_i)$$

As demonstrações dos lemas e teoremas a seguir podem ser vistas detalhadamente em (RONCHI, 2017, p. 23) [15] e (SHWARTZ; DAVID, 2014, p. 191) [16].

Temos então o primeiro lema sobre a convergência dos valores de β.

Lema 3.2. *Sejam $d^{(0)}, d^{(1)}, \ldots, d^{(T)}$ uma sequência arbitrária de vetores. Qualquer algoritmo com inicialização $\beta^{(0)} = 0$ e uma atualização da forma $\beta^{(k+1)} = \beta^{(k)} - \gamma \cdot d^{(k)}$, satisfaz*

$$\sum_{k=0}^{T} \langle \beta^{(k)} - \beta^*, d^{(k)} \rangle \leq \frac{\| \beta^* \|^2}{2\gamma} + \frac{\gamma}{2} \sum_{k=1}^{T} \| d^{(k)} \|^2 . \tag{10}$$

Em particular, para todo $B, L > 0$, se para todo k, $\| d^{(k)} \| \leq L$ e $\gamma = \sqrt{\frac{B^2}{L^2 T}}$, então para todo β^* com $\| \beta^* \| \leq B$,

$$\frac{1}{T} \sum_{k=0}^{T} \langle \beta^{(k)} - \beta^*, d^{(k)} \rangle \leq \frac{BL}{\sqrt{T}} \tag{11}$$

Que induz a minimização da função pelo teorema a seguir.

Teorema 3.2. *Sejam $B, L > 0$. Seja f uma função convexa e $\beta^* \in \arg\min_{\|\beta\| \leq B} f(\beta)$. Assuma que o método do gradiente estocástico rode T iterações com $\gamma = \sqrt{\frac{B^2}{L^2 T}}$, e que para todo k, $\| d^{(k)} \| \leq L$ com probabilidade 1. Então,*

$$\mathbb{E}[f(\overline{\beta})] - f(\beta^*) \leq \frac{BL}{\sqrt{T}}.$$

Além disso, para qualquer $\epsilon > 0$, para atingir $\mathbb{E}[f(\overline{\beta})] - f(\beta^*) \leq \epsilon$, basta executar o algoritmo do Gradiente Estocástico para um número de iterações T que satisfaça

$$T \geq \frac{B^2 L^2}{\epsilon^2}$$

3.4 O problema do gradiente estocástico

Mesmo sanando o problema da quantidade de erros que necessitam ser avaliados para uma única atualização do vetor β, o método do Gradiente Estocástico acaba trazendo consigo outro problema na atualização do modelo: muita variabilidade.

Tendo em vista que a atualização do vetor β muda a direção da minimização da função objetivo, na utilização deste método, cada erro entre o valor avaliado e o valor real acaba apresentando uma direção diferente para a minimização. Este fato pode acarretar com que, no trajeto de minimização, sejam realizados alguns desvios devido a direções diferenciadas que são tomadas, o que podemos chamar de passeios. Tais passeios acabam custando mais iterações do que seriam necessárias.

É possível compreender esta variabilidade com mais facilidade considerando as imagens comparativas dos métodos. Considerando um conjunto com apenas 25 pontos, podemos observar inicialmente o processo de Gradiente Descendente.

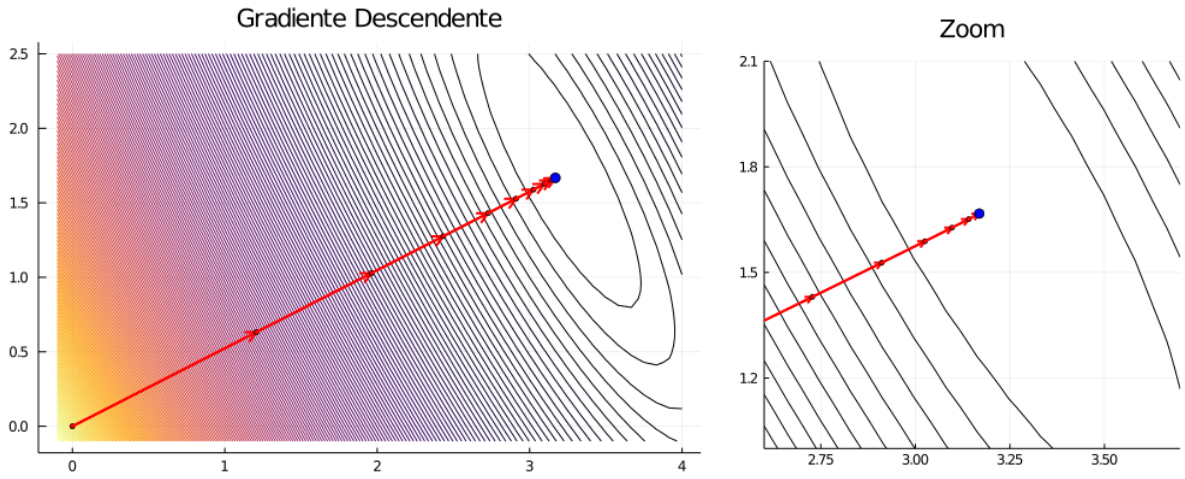


Figura 16: Caminho de otimização do Gradiente Descendente à esquerda e o zoom próximo a convergência à direita.

Agora, no mesmo conjunto de 25 pontos, realizamos o processo de otimização com o Método do Gradiente Estocástico.

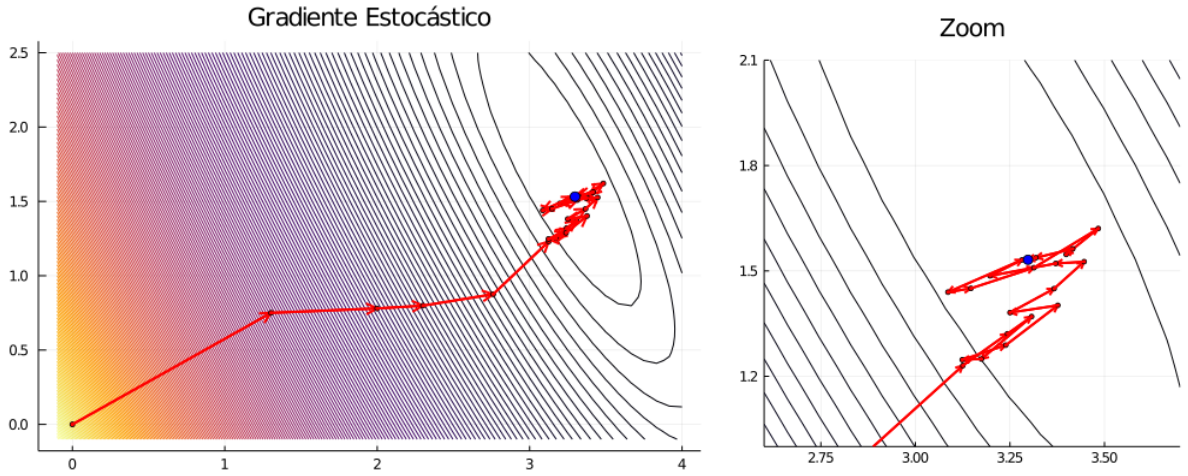


Figura 17: Caminho de otimização do Gradiente Estocástico à esquerda e o zoom próximo à convergência a direita.

Com o zoom das curvas de nível, fica bastante evidente a diferença entre os dois métodos, principalmente na região mais próxima ao ponto de convergência. Como mencionado, enquanto o Método do Gradiente Descendente traça o caminho com passos maiores e apontando diretamente ao ponto de minimização, o Método do Gradiente Estocástico, por alterar a direção a cada dado avaliado, progride através de pequenos passos em diversas direções diferentes até chegar em um possível mínimo. Além disso, é possível notar os passeios, problemas do Gradiente Estocástico que ocorrem próximo ao minimizador: o passo do algoritmo bastante pequeno e ainda caminha em diversas direções dispersas quanto mais próximo está do objetivo.

3.5 Gradiente estocástico *mini-batch*

Levando em consideração os problemas que ocorrem nos dois métodos apresentados, uma terceira opção surge visando atenuá-los simultaneamente.

Inicialmente foi apresentado o método do Gradiente Descendente que trazia consigo a necessidade de a cada iteração avaliar o banco de dados por completo para a atualização dos coeficientes, sendo custoso e muitas vezes demorado. Para desviar desta situação, o Gradiente Estocástico atualizava os coeficientes com a avaliação de cada dado, sendo assim, cada iteração envolvia apenas a avaliação de um resultado apenas, desta forma não se tem a demora do Gradiente Descendente porém, tem-se o problema da variabilidade das atualizações, pois cada erro determina uma direção diferente de atualização.

Ou seja, o melhor dos casos pode ser considerado quando se avalia não o banco de dados por completo e nem quando avalia-se dado a dado. Para este meio termo introduz-se o Método do Gradiente Estocástico com *mini-batch*.

A aplicação deste método consiste em pegar fatias aleatórias do conjunto de dados, avaliar o erro destas, aplicar o gradiente e atualizar os coeficientes. Funciona como se

fossem empregados "mini gradientes descendentes" em subconjuntos aleatórios do conjunto de dados total. A estes subconjuntos aleatórios atribuímos o nome de *mini-batch*, em que o tamanho do *batch* é escolhido empiricamente no momento de aplicação do método.

Por exemplo, considere que tem-se um conjunto com 1000 dados e suponha que o *batch* escolhido tem tamanho 100. Sendo assim, o algoritmo sorteará 100 dados aleatórios dentre os 1000, avaliará o erro destes e atualizará os coeficientes. Em seguida, tomará outros 100 dentre os que não foram avaliados ainda e repetirá o processo. No caso, faz-se isso 10 vezes e então se começa novamente o loop até que seja atingido algum dos critérios de parada.

Nem sempre o número de dados é divisível pelo valor do *batch* escolhido, sendo assim é necessário decidir o que fazer caso a divisão não seja inteira. No método aqui implementado, as fatias do conjunto de dados são tomadas com o tamanho do *batch*, caso haja resto na divisão, o último *batch* tem o tamanho deste resto. Por exemplo, com 11 dados e *batch* de tamanho 2, são realizados 5 atualizações de tamanho 2 e uma de tamanho 1.

Com isso tudo em vista, considere $X \in \mathbb{R}^{n\times p}$, n_b como tamanho do *batch*, a função h_β que se ajusta ao conjunto de dados, a função ℓ que avalia o erro do modelo e definimos

$$L_i(\beta) = \ell(y^{(i)}, h_\beta(x^{(i)})),$$

podemos escrever o algoritmo da seguinte forma:

Algoritmo 3: Método do Gradiente Estocástico *mini-batch*

Dados: Escolha $\beta^{(0)} \in \mathbb{R}^p, \gamma_k \in \mathbb{R}^*_+, K \in \mathbb{N}, n_b \in \mathbb{N}$ e J conjunto de índices aleatórios entre 1 e n;

para $k = 1, \ldots, K$ **faça**
  Tome $JS = \{J_1, J_2, \ldots, J_N\}$, subconjuntos aleatórios de J com $N = \lceil \frac{n}{n_b} \rceil$
  para $J_j \in JS$ **faça**
    $d^k = \sum_{i\in J_j} \nabla L_i(\beta)/|J_j|$
    $\beta^{(k+1)} \leftarrow \beta^{(k)} - \gamma_k \cdot d^{(k)}$;
  fim
fim
retorna $\beta^{(K)}$

Para ter uma noção mais visual do mini-batch em funcionamento, pode-se utilizar o mesmo conjunto de 25 dados que foi apresentado anteriormente utilizando Gradiente Descendente e Estocástico. Neste, aplicamos o *batch* de tamanho 5.

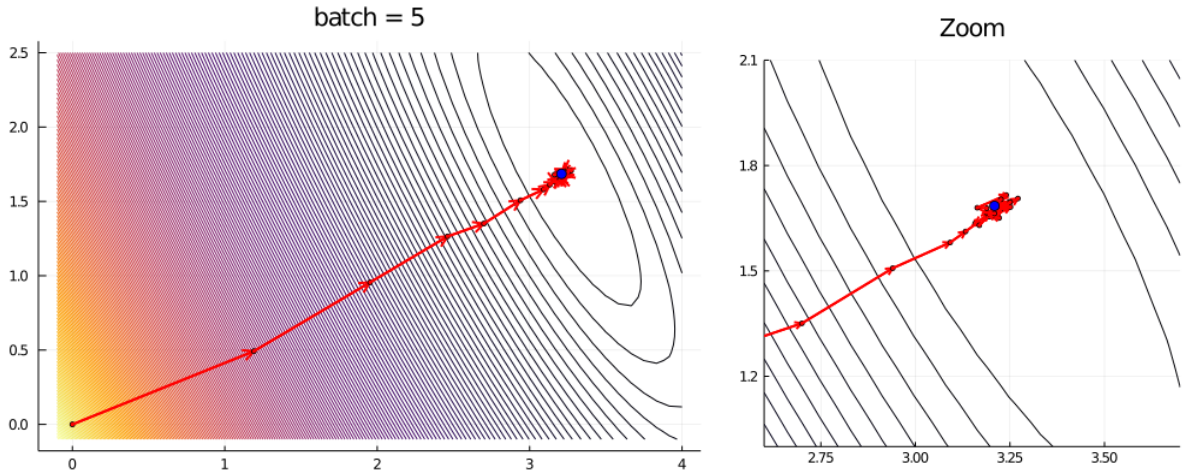


Figura 18: Caminho de otimização do Gradiente Estocástico com $batch = 5$ à esquerda e o zoom próximo à convergência à direita.

Note que com este aumento do tamanho de *batch*, a trajetória já não tem tantos desvios quanto era possível se ver com o Gradiente Estocástico puro, isto é, $batch = 1$, o que já é uma ótima melhora ao caminho de otimização se tornando até de certa forma semelhante ao caminho obtido com o Gradiente Descendente puro, ou seja, tamanho do *batch* igual à quantidade total de dados do conjunto.

Vale também ressaltar que, tanto quanto sabemos, não existe um tamanho definitivo de *bacth* que define os melhores resultados. Este é decidido de acordo com o problema a ser resolvido, levando em conta o tempo disponível para a aplicação do método, a precisão necessária da resolução, o poder computacional disponível, entre outros fatores.

3.6 Regularização

Na fase de treinamento do modelo, como apontado anteriormente, um problema que deve ser evitado é o *overffiting*. Este ajuste excessivo no conjunto de dados de treino pode trazer péssimos resultados quando o modelo for colocado em prática nos dados de teste. Tendo isso em vista, ao invés de apenas minimizar a função erro, é possível acrescentar uma parcela a mais no modelo, que visa melhorar sua regularidade e diminuir a complexidade do mesmo no conjunto de treinamento. Esta parcela é conhecida como termo de regularização.

Sendo assim, considere uma função de erro dada por $L(\beta)$, o modelo para o treinamento agora passa a ser escrito como

$$\min_{\beta \in \mathbb{R}^p} L(\beta) + \alpha \cdot R(\beta), \tag{12}$$

em que $R(\beta)$ é a função de regularização do modelo para os coeficientes em β e a constante α é o hiperparâmetro que recebe o nome de parâmetro de regularização. Esta constante

pode ser interpretada como a responsável pela força da influência do termo de regularização no modelo sendo treinado, ou seja, quanto maior for α, menor será a influência sobre parâmetro β e quanto menor for α, maior será a influência sobre o parâmetro β.

Para a função $R(\beta)$ existem três possibilidades tradicionais que serão implementadas no modelo do trabalho. A escolha da função de regularização é dada de acordo com o objetivo do problema escolhido.

- **Regularização LASSO ou ℓ_1**

$$R(\beta) = \sum_{j=0}^{p} |\beta_j|$$

  Esta função tende a diminuir a variância do modelo. Além disso, quando existem mais que uma características correlacionadas, isto é, características que acabam se comportando de maneira muito semelhante no modelo, esta função tende a selecionar apenas uma delas e zera o coeficiente das outras.

- **Regularização Ridge ou ℓ_2**

$$R(\beta) = \frac{1}{2} \sum_{j=0}^{p} \beta_j^2 = \frac{1}{2} \parallel \beta \parallel_2^2$$

  Esta regularização, diferente da anterior, é diferenciável, o que traz continuidade às penalizações. Além disso, a continuidade faz com que todas as características sejam levadas em consideração na minimização, o que pode ser bom pela consistência dos ajustes.

- **Regularização Elasticnet**

$$R(\beta) = \rho \frac{1}{2} \sum_{j=0}^{p} \beta_j^2 + (1 - \rho) \sum_{j=0}^{p} |\beta_j|$$

  Para esta função, temos apenas a combinação convexa das funções utilizadas no ℓ_1 e ℓ_2, sendo assim pode apresentar as qualidades de ambas. Aqui porém deve ser considerada esta constante $\rho \in [0, 1]$, um hiperparâmetro que deve ser ajustado para se obter o melhor resultado possível para o processo de minimização, o que nem sempre pode ser um processo simples ou prático.

Uma das grandes necessidades da utilização do termo de regularização pode ficar bastante explicita no contexto da Regressão Logística apresentada anteriormente.

Esta função tem como motivação a previsão de classificação binária de dados futuros, para isso, utilizamos com a função os coeficientes β_i, que auxiliam na função ajustando-a com mais perícia aos dados binários.

Porém, podemos analisar o simples problema com dois pontos a seguir na Figura 19.

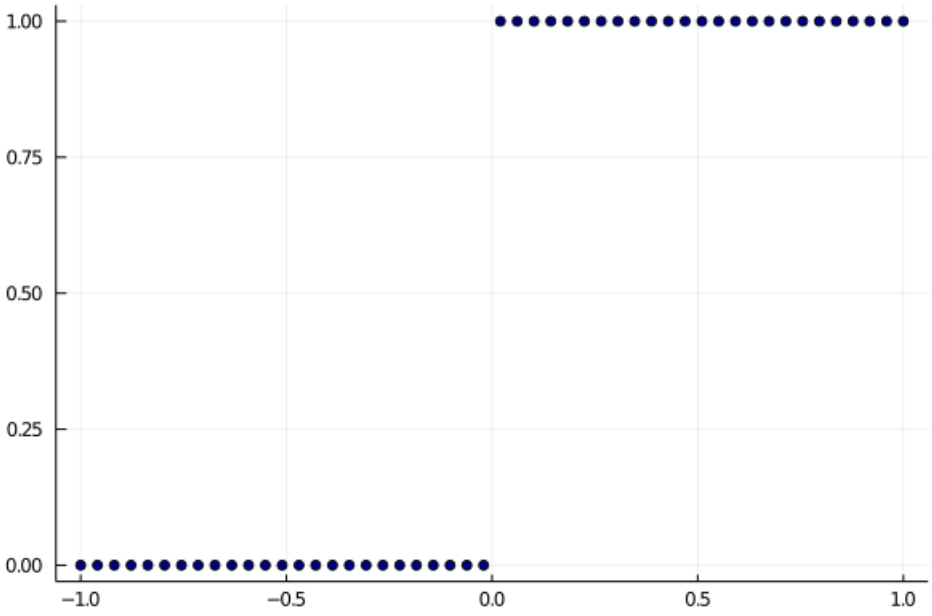


Figura 19: Regressão Logística com dados perfeitamente separáveis.

Note que, muito facilmente, tomando a reta $x = 0$ já temos a separação completa dos dois pontos, não sendo necessária a aplicação da sigmoide. Porém, a criação do modelo de Regressão Logística deve conseguir abranger a maior variedade de situações possíveis em conjuntos de dados, sendo assim, vale analisar o que aconteceria com o ajuste do modelo $h_\beta(x) = \sigma(\beta_0 + \beta_1 x)$ a este caso.

Como já apresentado na Seção 2, para este modelo, enquanto β_0 é o responsável pelo deslocamento do gráfico pelo eixo x, β_1 é o responsável pela inclinação da sigmoide, seu esticamento e encolhimento, sendo quanto maior o β_1, maior a inclinação da função, como pode ser visto a seguir na Figura 20

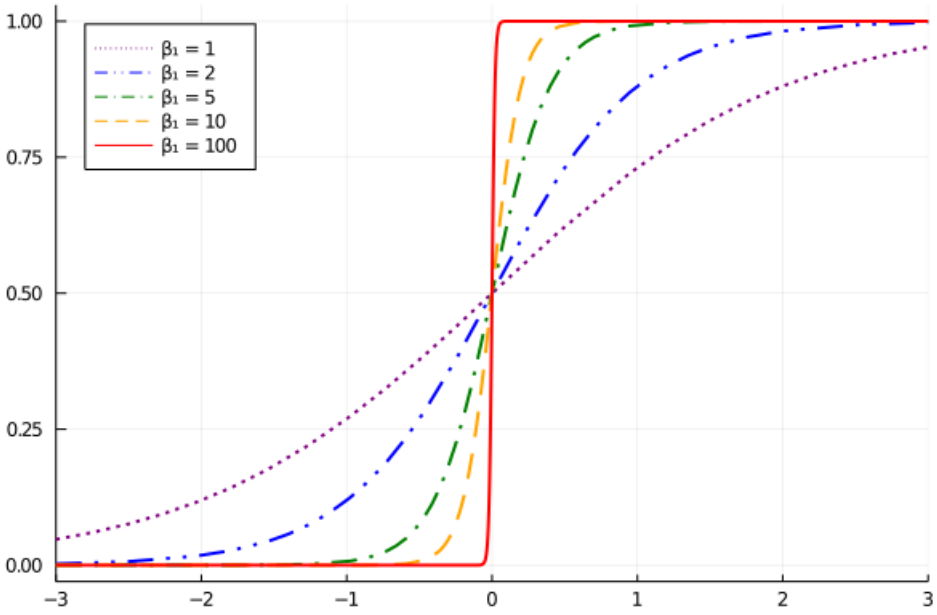


Figura 20: Inclinação da *sigmoide* para grandes valores de β_1.

Note que para a separação dos pontos apresentados, β_1 poderia adotar um valor bastante elevado, forçando a curva a ficar muito semelhante à função degrau na Figura 21.

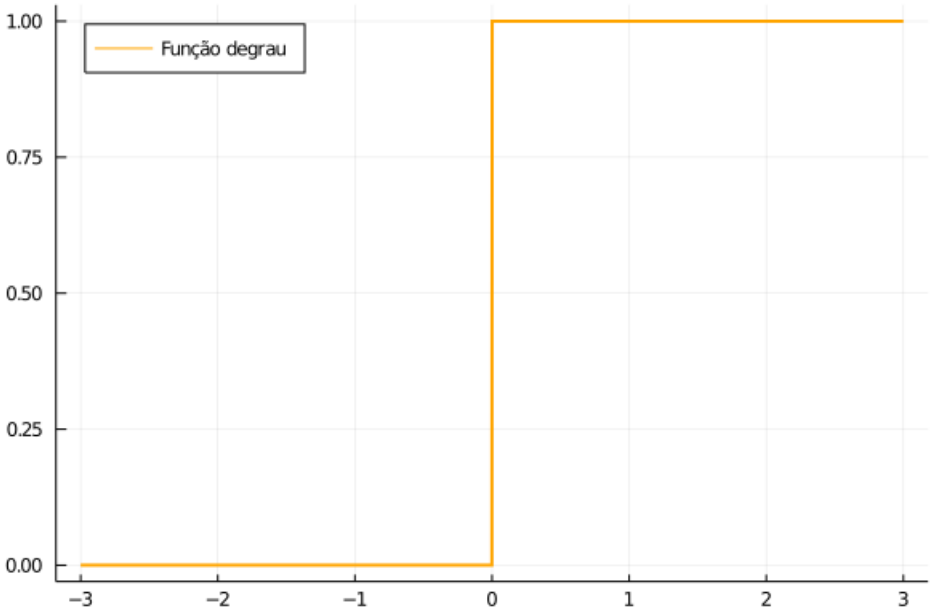


Figura 21: Gráfico da função degrau.

Porém, para este conjunto de dados em específico, com qualquer *sigmoide* tomada já é possível de realizar a separação, simplesmente rotulando valores $h_\beta(x) > 0{,}5$ como 1 e 0 caso contrário. Este processo ocorre pois o conjunto de dados em questão é um conjunto perfeitamente separável. Sendo assim, em casos como esse, como qualquer valor para β_1

satisfaz a separação, se torna interessante a aplicação de alguma forma de "freio" para que os coeficientes β não cresçam de maneira exagerada.

Esta situação citada é o que nos induz a adicionar, ao modelo de Regressão Logística, alguma das funções $R(\beta)$ citadas anteriormente, resultando no problema

$$\min_{\beta \in \mathbb{R}^p} -\sum_{i=1}^{n} \left[y^{(i)} \log(h_\beta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\beta(x^{(i)})) \right] + \alpha R(\beta),$$

em que, simultaneamente a minimização do erro do modelo, visa-se a minimização dos valores adotados por β.

4 Implementação e Resultados

Todo o estudo realizado previamente tinha como objetivo principal a implementação do Método do Gradiente Estocástico e de alguns modelos matemáticos de Aprendizagem de Máquina no pacote MultObjNLPModels na linguagem de programação Julia.

4.1 O pacote MultObjNLPModels

Este pacote surge com a ideia de estender os conteúdos do pacote NLPModels, uma biblioteca de resolução de problema de otimização não-linear. Essa tem como objetivo a representação e solução de problemas de Otimização que podem ser escritos da forma geral como

$$\begin{aligned} \min \quad & f(x) \\ \text{sujeita a} \quad & c_i(x) = 0, \quad i \in E, \\ & c_{L_i} \leq c_i(x) \leq c_{U_i}, \quad i \in I, \\ & \ell \leq x \leq u, \end{aligned}$$

onde, $f : \mathbb{R}^n \to \mathbb{R}$, $c : \mathbb{R}^n \to \mathbb{R}^m$, $E \cup I = \{1, 2, \ldots, m\}$, $E \cap I = \emptyset$, e $c_{L_i}, c_U, \ell_j, u_j \in \mathbb{R} \cup \{\pm\infty\}$ para $i = 1, \ldots, m$ e $j = 1, \ldots, n$.

Como é possível notar, o problema consistia em uma única função objetivo f. A ideia de estender esse pacote, como citado, consiste em considerar mais que uma função objetivo, ou seja, tomar um problema da forma

$$\begin{aligned} \min \quad & \sum_i \sigma_i f_i(x) \\ \text{sujeita a} \quad & c_i(x) = 0, \quad i \in E, \\ & c_{L_i} \leq c_i(x) \leq c_{U_i}, \quad i \in I, \\ & \ell \leq x \leq u, \end{aligned}$$

em que, $c, c_{L_i}, c_U, \ell_j, u_j$ são análogos ao caso anterior mas, além destes, consideramos, $f_i : \mathbb{R}^n \to \mathbb{R}$ e σ_i como a função objetivo sob i e seu respectivo peso. Ou seja, para este pacote considera-se a minimização de uma combinação linear de diversas funções objetivos.

Com este modelo de função objetivo mais geral, torna-se possível a implementação de forma mais prática de alguns algoritmos e modelos tradicionais da Aprendizagem de Máquina, pois consiste essencialmente da generalização apresentada ao final da Seção 2, dada por

$$L(\beta) = \frac{1}{n} \sum_{i=1}^{n} L_i(\beta), \quad \text{onde} \quad L_i(\beta) = \ell(y^{(i)}, h_\beta(x^{(i)})),$$

sendo h_β o modelo matemático a ser ajustado e ℓ a função que mede o erro.

4.2 A estrutura RegressionModels

RegressionModel é uma estrutura em linguagem Julia [1] que consiste essencialmente de uma matriz X, um vetor y, uma função $h(x, \beta)$ e uma função $\ell(y, \hat{y})$. Com essa estrutura genérica, o processo do modelo consiste em avaliar a função $h(x, \beta)$ nas linhas da matriz X, obtendo $\hat{y} = h(X, \beta)$. E então calcular o erro dado por $\ell(y, \hat{y})$.

Como foi generalizado na Seção 2, todos os modelos de Aprendizagem de Máquina aqui aplicados podem ser escritos de uma maneira muito semelhante a esta estrutura. Isso então permite que sejam construídas as funções *LinearRegressionModel* e *LogisticRegressionModel* que criam um modelo *RegressionModel* utilizando $h(x, \beta)$ e $\ell(y, \hat{y})$ específicas.

Portanto, ao chamar a função *LinearRegressionModel*, ela constrói um *RegressionModel* definindo

$$h(x, \beta) = h_\beta(x) = x^T \beta$$

e

$$\ell(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2.$$

Analogamente, com a chamada da função *LogisticRegressionModel*, é construído um *RegressionModel* que consiste das funções

$$h(x, \beta) = h_\beta(x) = \frac{1}{1 + e^{-x^T \beta}}$$

e

$$\ell(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}).$$

4.3 Implementação do gradiente estocástico

Com as estruturas construídas no *RegressionModels*, para a solução do problema, ou seja, encontrar o valor para β que minimize a soma dos erros avaliados, é necessária então a implementação do Método do Gradiente Estocástico.

Sendo assim, analisemos o método do Gradiente Estocástico apresentado no Algoritmo 1, ou seja, a atualização do β dada apenas por

$$\beta^{(k+1)} = \beta^{(k)} - \gamma_k \cdot \nabla L_i(\beta), \qquad \forall i = 1, \ldots, n.$$

Com esta primeira formulação já é possível obter resultados bastantes satisfatórios, então o passo a partir daqui consistem apenas em aplicar alterações que consigam otimizar o processo de atualização do β.

A primeira alteração se trata do problema existente quando o conjunto de dados é perfeitamente separável, que como apresentado na Seção 3, acaba forçando alguns dos coeficientes β_i do modelo tenderem para infinito. Sendo assim, é acrescentado o termo de regularização $\alpha \cdot R(\beta)$ para ser minimizado juntamente com $L_i(\beta)$, ou seja, obtemos

$$\beta^{(k+1)} = \beta^{(k)} - \gamma_k \cdot \nabla \left[L_i(\beta) + \alpha \cdot R(\beta) \right], \quad \forall i = 1, \ldots, n.$$

Lembrando que, a função $L_i(\beta)$ é dependente das funções ℓ e h_β e estas são diferentes dependendo do modelo aplicado, ou seja, utilizando *LinearRegressionModel* são aplicadas funções ℓ e h_β diferentes de quando é utilizado *LogisticRegressionModel*. Mas, a função de regularização tem que apenas três opções diferentes que podem ser escolhidas no momento de aplicação do método, independentemente do modelo escolhido. Sendo assim, ao invés de implementar a função $R(\beta)$ e gastar poder computacional para calcular sua derivada a cada iteração, faz sentido implementar no código diretamente as derivadas da função $R(\beta)$.

Fazendo isso então, podemos analisar cada um dos casos, começando com o mais simples: Regularização Ridge ou ℓ_2 que, como é diferenciável, temos

$$R(\beta) = \frac{1}{2} \sum_{j=0}^{p} \beta_j^2 \quad \Longrightarrow \quad \frac{\partial R}{\partial \beta_j}(\beta) = \beta_j, \quad \forall j = 0, \ldots, p.$$

Ou seja, na implementação do código, definimos que a parcela $\nabla_i R(\beta)$ é escrita apenas como β_i, basicamente uma identidade.

Para a segunda opção de $R(\beta)$, a regularização do tipo LASSO ou ℓ_1, é necessário analisar com mais cautela. Lembrando que esta é dada por

$$R(\beta) = \sum_{j=0}^{p} |\beta_j|.$$

Note que,

$$\beta_i > 0 \quad \Longrightarrow \quad \frac{\partial R}{\partial \beta_i}(\beta) = 1$$

$$\beta_i < 0 \quad \Longrightarrow \quad \frac{\partial R}{\partial \beta_i}(\beta) = -1.$$

Ainda, a função não é diferenciável para $\beta_i = 0$, o que torna necessário definir a derivada nestes ponto com algum valor. Tomando então como 0, é possível definir a derivada de R da como a função sinal de β, ou seja,

$$R(\beta) = \sum_{i=0}^{p} |\beta_j| \quad \Longrightarrow \quad \frac{\partial R}{\partial \beta_j}(\beta) = sgn(\beta_j), \quad \forall j = 0, \ldots, p,$$

pois

$$sgn(\beta_j) = \begin{cases} 1, & se \quad \beta_j > 0 \\ 0, & se \quad \beta_j = 0 \\ -1, & se \quad \beta_j < 0. \end{cases}$$

Com estes dois casos definidos, pode-se facilmente definir a derivada para o caso Elasticnet da forma

$$R(\beta) = \rho \frac{1}{2} \sum_{j=0}^{p} \beta_j^2 + (1-\rho) \sum_{j=0}^{p} |\beta_j|$$

$$\Longrightarrow \quad \frac{\partial R}{\partial \beta_j}(\beta) = \rho \cdot \beta_j + (1-\rho) \cdot sgn(\beta_j), \quad \forall j = 0, \ldots, p.$$

A segunda alteração a ser realizada é sobre o parâmetro γ_k. Este determina o tamanho do passo a cada iteração rumo à otimização do problema. Se este passo for muito grande, o processo de minimização pode passar direto pelo ponto ótimo e, se tomado muito pequeno, pode exigir iterações excessivas até o ponto ótimo. Sendo assim, é necessário encontrar um tamanho adequado ao passo de modo que o processo de otimização seja satisfatório.

Como mencionado anteriormente, o passo pode ser tomado constante, sendo $\gamma_k = 10^{-2}$ um valor que apresentou resultados bastante interessantes nos dados utilizados. Porém, com o decorrer do processo de minimização da função, na medida em que se aproxima do objetivo, uma estratégia interessante é a diminuição do passo, para aumentar a precisão do método quanto mais próximo se estiver do ponto ótimo. Tendo esta ideia em vista,

existem algumas maneiras de implementar a mudança do passo ao decorrer das iterações.

Tomando como referencial o modelo apresentado pelo o pacote *scikit-learn* [13] em linguagem Python, dois passos diferentes foram implementados. O primeiro deles recebe o nome de *Inverse Scaling* e é definido como

$$\gamma_k = \frac{\gamma_0}{(k+1)^\nu},$$

onde k é o índice da iteração que começa em 0 e γ_0, ν são hiperparâmetros, escolhidos a partir de testes em conjuntos menores de dados sendo $\gamma_0 = 1$ e $\nu = 0{,}5$. Inicialmente, este passo se baseia apenas no inverso de cada iteração para minimizar o tamanho de cada passo, ou seja,

$$\gamma_k = \frac{\gamma_0}{k+1}$$

porém, de acordo com Bottou [2], considerando a não regularidade de um conjunto de dados, acaba-se sugerindo a utilização do expoente $\nu = 0{,}5$.

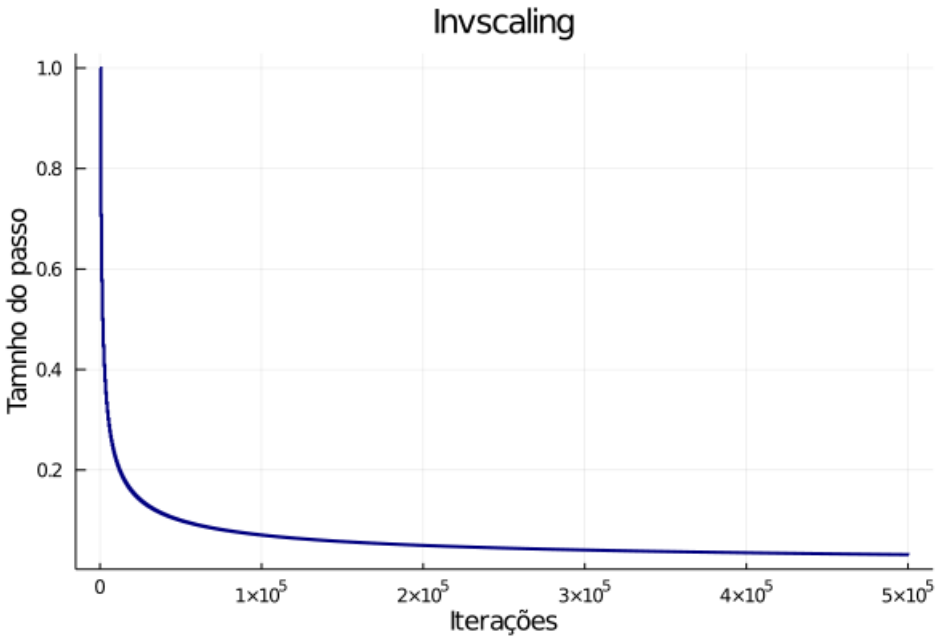


Figura 22: Decrescimento do valor do passo por *Invscaling*.

O segundo passo retirado do pacote *scikit-learn* é o que recebe o nome de *optimal*. Este passo também é mencionado de forma semelhante em [2]

$$\gamma_k = \frac{\gamma_0}{\alpha(t_o + k)},$$

em que k é o índice de iteração, t_0 é um hiperparâmetro determinado após alguns testes como 10^3, $\gamma_0 = 1$ é um valor inicial para o passo e $\alpha = 10^{-2}$ é o mesmo que parâmetro de regularização, o que talvez seja a maior diferença entre os outros passos aqui em

questão. Outro ponto importante a ressaltar sobre este passo é que ele já inicia com um tamanho muito pequeno, que dificulta a comparação direta com os outros mas que, quanto implementado, apresenta geralmente bons resultados.

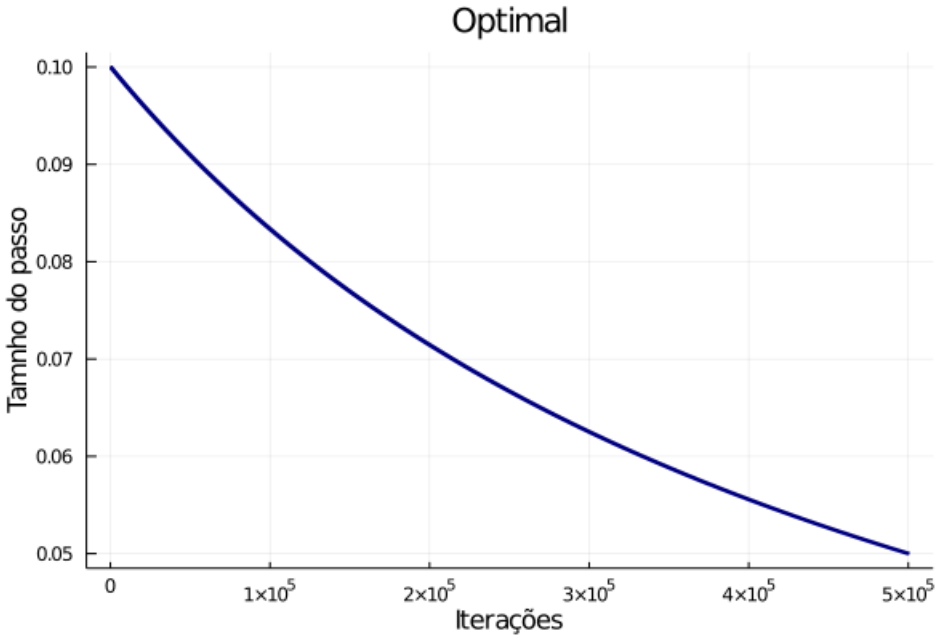


Figura 23: Decrescimento com o passo *optimal*.

Ainda utilizando como referência (BOTTOU, 2012, p. 10) [2], dadas algumas alterações algébricas no passo anterior, outro passo implementado é o que recebe o nome de *time-based*, definido como

$$\gamma_k = \frac{\gamma_0}{1 + dk},$$

em que k é o índice da iteração novamente, $\gamma_0 = 1$ é um valor inicial para o passo e $d = 0{,}5$ é um hiperparâmetro chamado de parâmetro de decaimento determinado a partir de alguns testes.

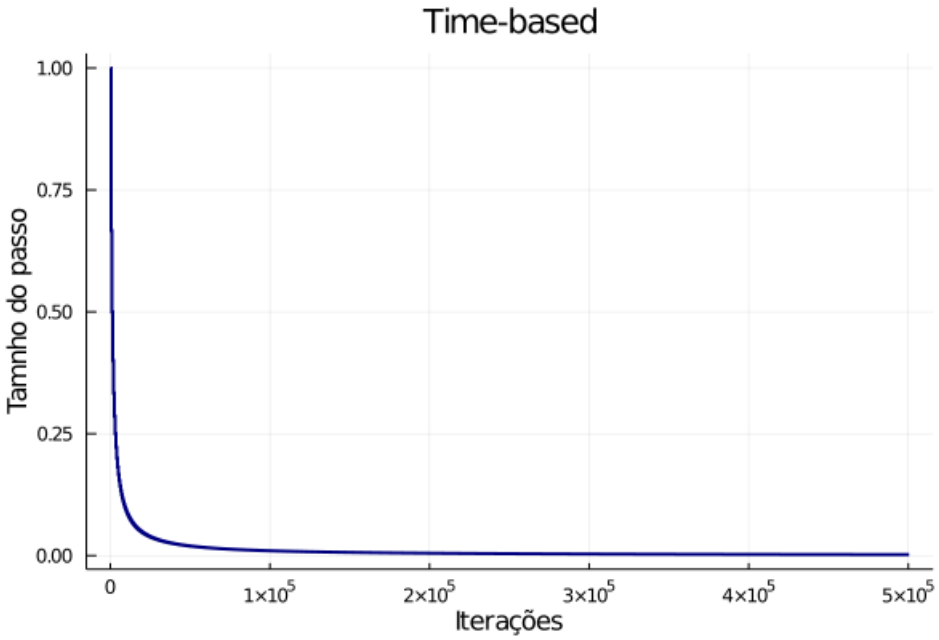


Figura 24: Decrescimento do passo dado por *time-based*.

Além destes dois passos, em [12] são sugeridos outros dois tipos de passos que apresentam bons resultados. O primeiro deles, tendo em vista o decaimento aparentemente exponencial dos anteriores, é definido como

$$\gamma_k = \frac{\gamma_0}{e^{dk}},$$

em que, novamente $\gamma_0 = 1$ é o passo inicial escolhido a partir de testes, $d = 0{,}5$ é o parâmetro de decaimento escolhido a partir de testes e k é o índice de iteração. Neste caso, o passo acaba diminuindo de maneira bastante rápida.

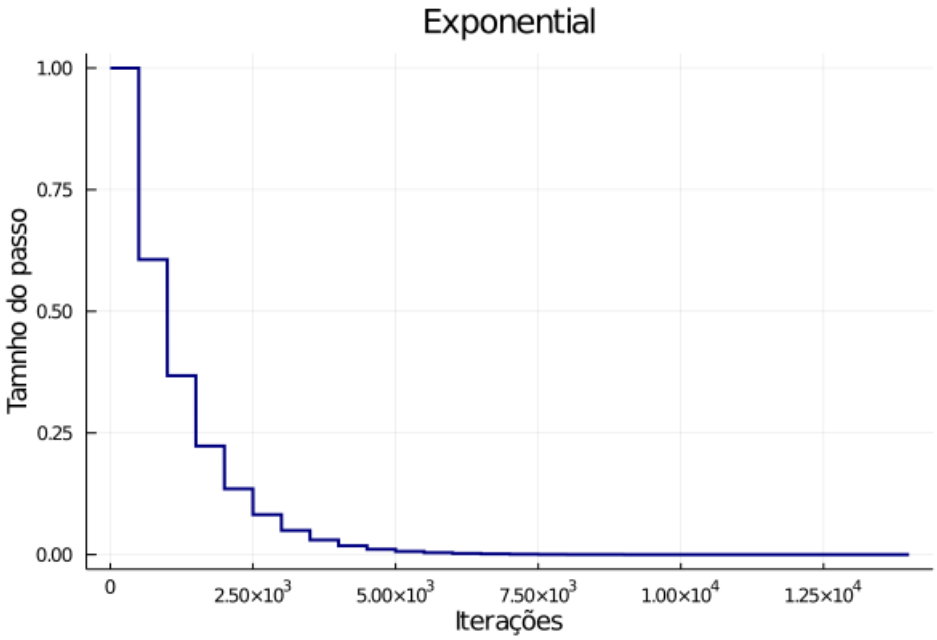


Figura 25: Decrescimento do passo exponencial.

Finalmente, o último passo, de nome *step-based*, definido como

$$\gamma_k = \gamma_0 d^{\lfloor \frac{k+1}{r} \rfloor},$$

sendo $\gamma_0 = 1$ o passo inicial, $d = 0{,}5$ o parâmetro de decaimento, k o índice de iteração e $r = 10$ também um hiperparâmetro entendido como a taxa de queda. Este, talvez sendo o passo mais diferente, tem seu valor alterado a cada r iterações do modelo, evitando possíveis diminuições excessivas entre um passo e outro.

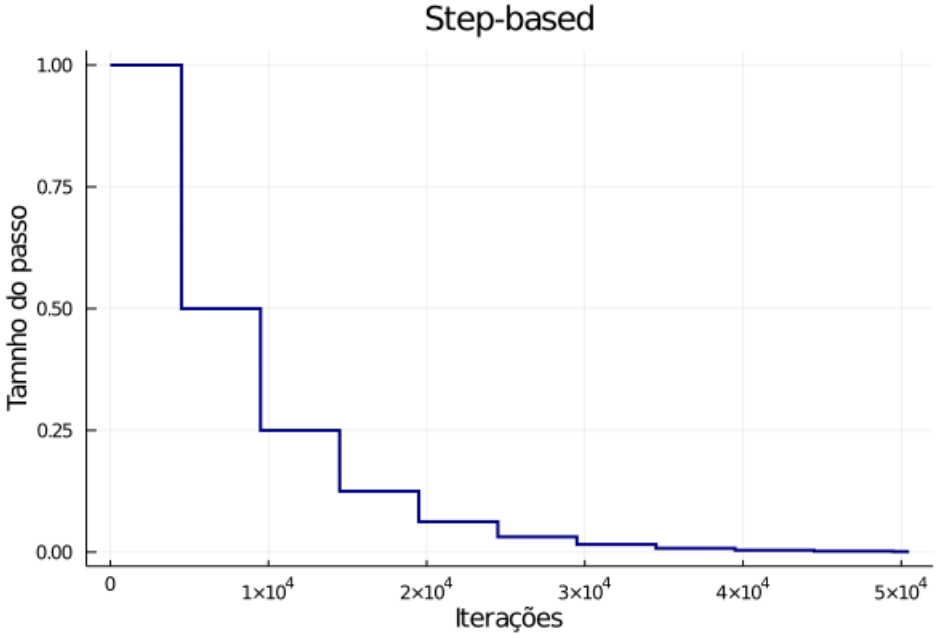


Figura 26: Decrescimento do passo *step-based*.

A escolha do passo a ser utilizado no momento da aplicação depende muito do conjunto de dados envolvido. Por exemplo, para problemas do tipo de Regressão Linear, o pacote *scikit-learn* [13] recomenda a utlização do passo *invscaling*. Sendo assim, faz sentido analisar todos os passos lado a lado para entender melhor a diferença entre eles.

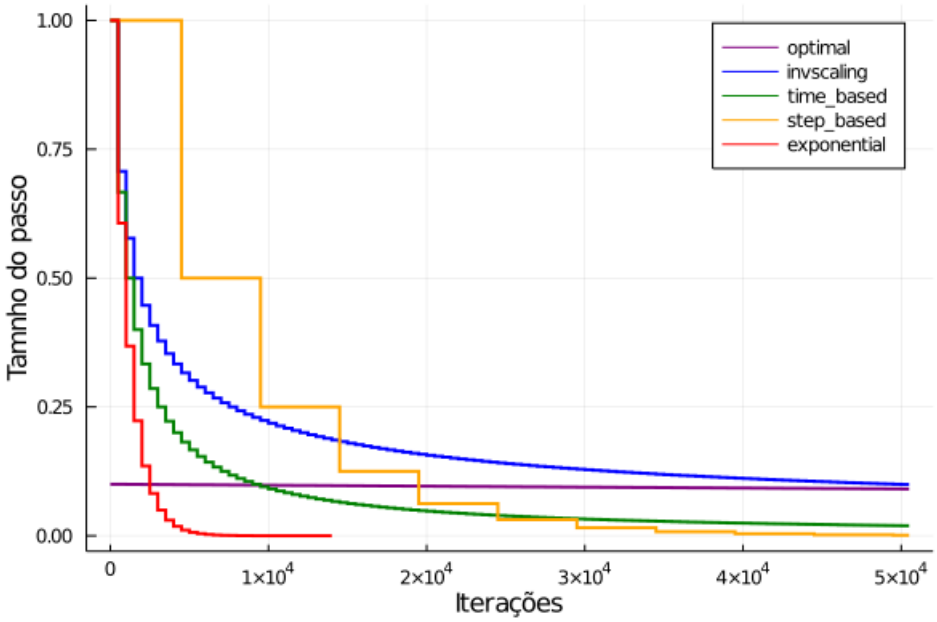


Figura 27: Decaimento de todos os passos implementados no pacote.

Note que, como o passo *optimal* já inicia muito pequeno, quando comparado aos outros diretamente aparentemente se torna uma linha reta devido à escala.

Com a todas as Regularizações e todas as diferentes opções de passos já implementados no código, outro ponto bastante importante a ser abordado sobre a implementação deste algoritmo de Otimização é a escolha dos critérios de parada do algoritmo.

Como mencionado na Seção 3, pela Proposição 3.1, o minimizador da função seria onde esta tem sua derivada nula, mas, considerando apenas este critério de parada, muitos problemas tendem a surgir, tal como o algoritmo encontrar um ponto cuja função tem um valor muito pequeno que ainda não é exatamente zero, porém é pequeno demais para ainda haver progressos significativos. Sendo assim, outros critérios de parada para o processo de minimização fazem-se extremamente necessários.

Os critérios aqui escolhidos consistem em:

- Tempo máximo de funcionamento de 60 segundos.
- Número máximo de iterações em 1000.
- Tamanho do passo γ_k se tornar menor que 10^{-6}.

Atingindo um destes critérios, o algoritmo encerra suas iterações e então entrega o valor de β encontrado até a iteração em questão. Como pode ser observado na Figura 4.3,

pela alta velocidade de decrescimento dos passos como *exponential* e *time_based*, esses têm uma maior tendência a fazer o algoritmo atingir o tamanho do passo como critério de parada.

4.4 Comparações

Finalizada a implementação do código e determinados os hiperparâmetos mais adequados para o modelo, foram realizados testes com conjuntos de 100 e 200 pontos aleatórios considerando diferentes graus de aleatoriedade dos mesmos. Para cada um destes conjuntos de pontos foram experimentados diferentes tamanhos de *batches* e, para cada tamanho de *batch* utilizado, o teste foi realizado 1000 vezes.

Modelo Linear

Iniciando os testes com o modelo linear, ou seja, a função *LinearRegressionModel*, para a avaliação do resíduo obtido de acordo com o tamanho de *batch*, utilizou-se a função dada por

$$r(y, \hat{y}) = \frac{1}{2} \parallel y - \hat{y} \parallel^2,$$

onde y consiste nos valores reais e $\hat{y}$ nos valores previstos pelo modelo.

Para a criação do conjunto de pontos para este caso, considerou-se pontos distribuídos em torno de uma reta, sendo dados então da seguinte forma:

$$y = 2x + 3 + \varepsilon, \quad \varepsilon \sim N(0; 0{,}4),$$

considerando que x são pontos aleatórios entre 0 e 1, podemos visualizar este conjunto de dados na Figura 28 a seguir.

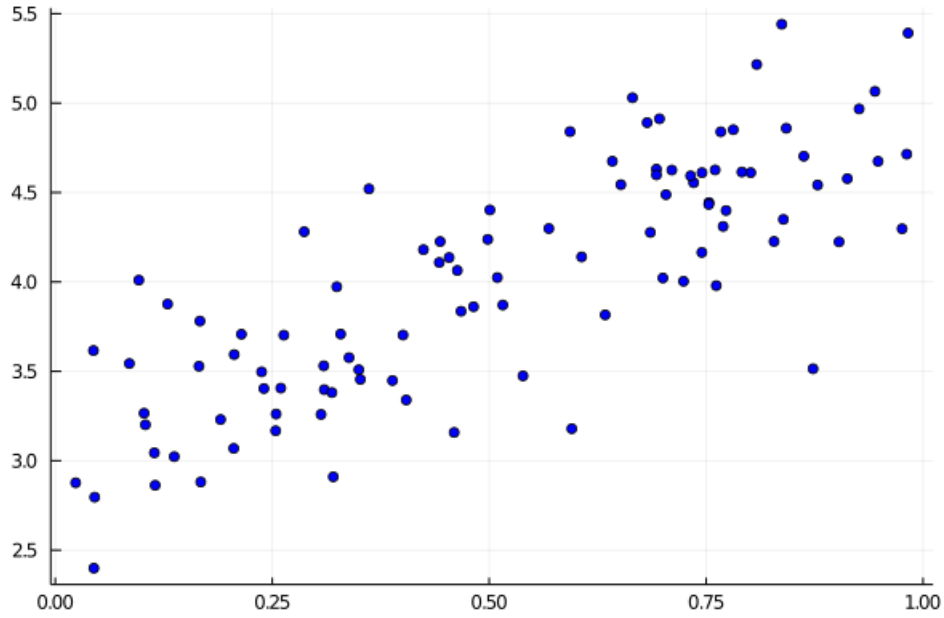


Figura 28: Amostra de 100 pontos do conjunto de pontos lineares.

Considerando então inicialmente apenas 100 pontos distribuídos desta forma, tomou-se *batches* dos tamanhos 1, 2, 5, 10, 50, 100. Para cada um destes tamanhos, o algoritmo foi executado por completo 1000 vezes e a cada uma delas foi calculado o resíduo. Com estes valores é possível construir o diagrama de dispersão na Figura 29.

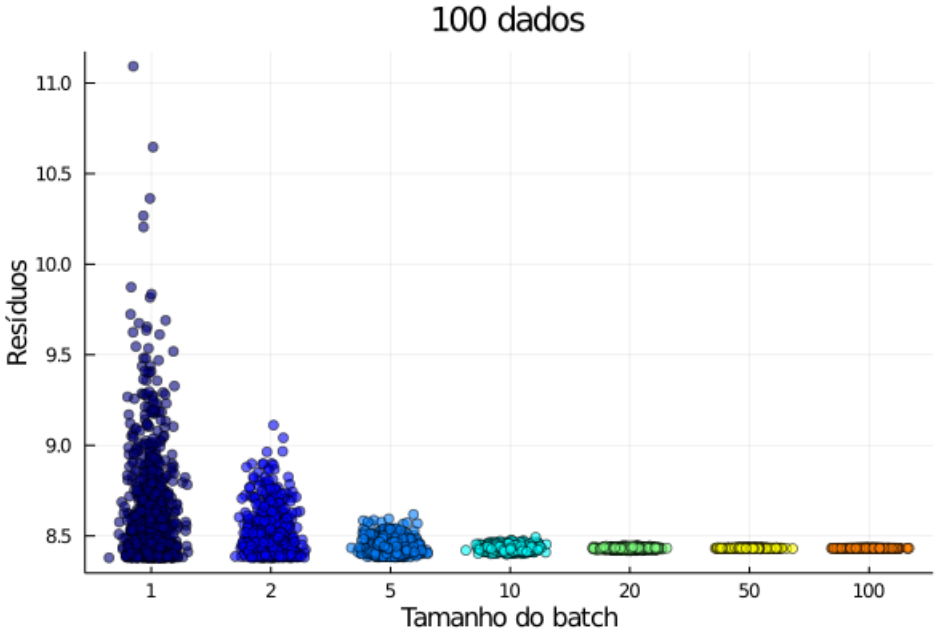


Figura 29: Dispersão do modelo linear com 100 pontos.

Note que os passeios realizados no Gradiente Estocástico isto é, $batch = 1$, fazem com o que o modelo tenha bastante variabilidade nas respostas, determinando resíduos

diferentes. Porém, conforme se aumenta o tamanho do *batch* até o ponto de se obter o método do Gradiente Descendente, esta variabilidade vai diminuindo pois, como pode ser visto na Figura 16, a direção utilizada apresenta melhor desempenho.

O padrão de dispersão de resíduos com o conjunto de 100 dados se mantém quando são considerados os conjuntos de 200 dados, que pode ser observado na Figura 30.

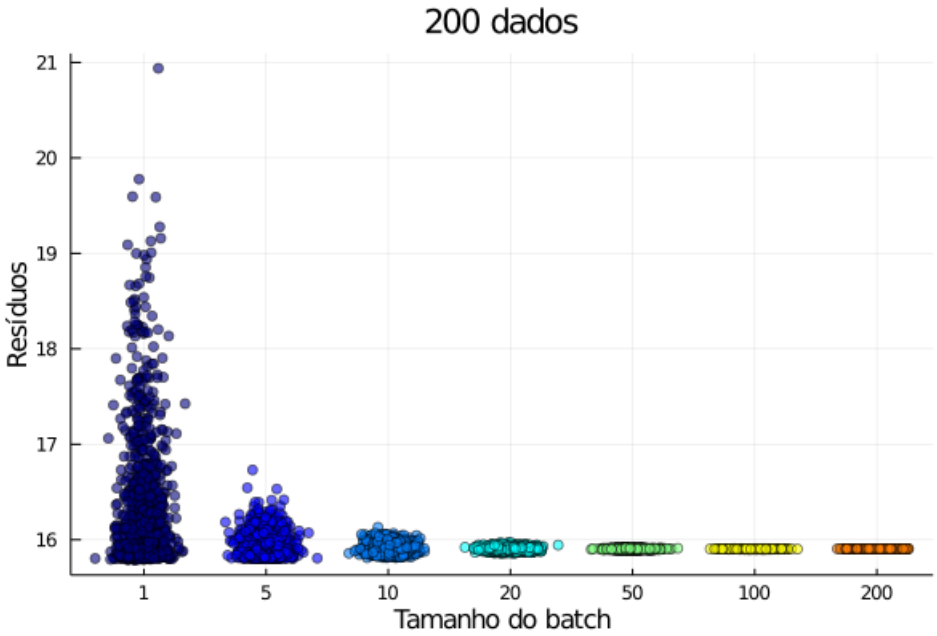


Figura 30: Dispersão do modelo linear com 200 pontos.

Outra maneira de observar os efeitos da mudança de *batch* nos resíduos pode ser considerando o resíduo ótimo.

Tendo em vista que o problema aqui tratado consiste em um modelo linear, pode-se obter um valor de β ótimo a partir da resolução do sistema dado por

$$X\beta = y \quad \Longrightarrow \quad X^T X\beta_{otim} = X^T y.$$

Com o valor de β_{otim} determinado, obtemos

$$R_{otim} = y - X\beta_{otim},$$

o resíduo ótimo. Onde y seriam os valores originais e $X\beta_{otim}$ consiste na previsão obtida pelo β_{otim}. Com este valor, é possível plotar o gráfico dado pela diferença do resíduo obtido a cada iteração com o resíduo ótimo R_{otim}. Considerando o passo *time-based* com $\alpha = 10^{-6}$, $\gamma_0 = 0{,}1$ e $d = 0{,}1$ obtemos então, nas Figuras 31, 32 e 33.

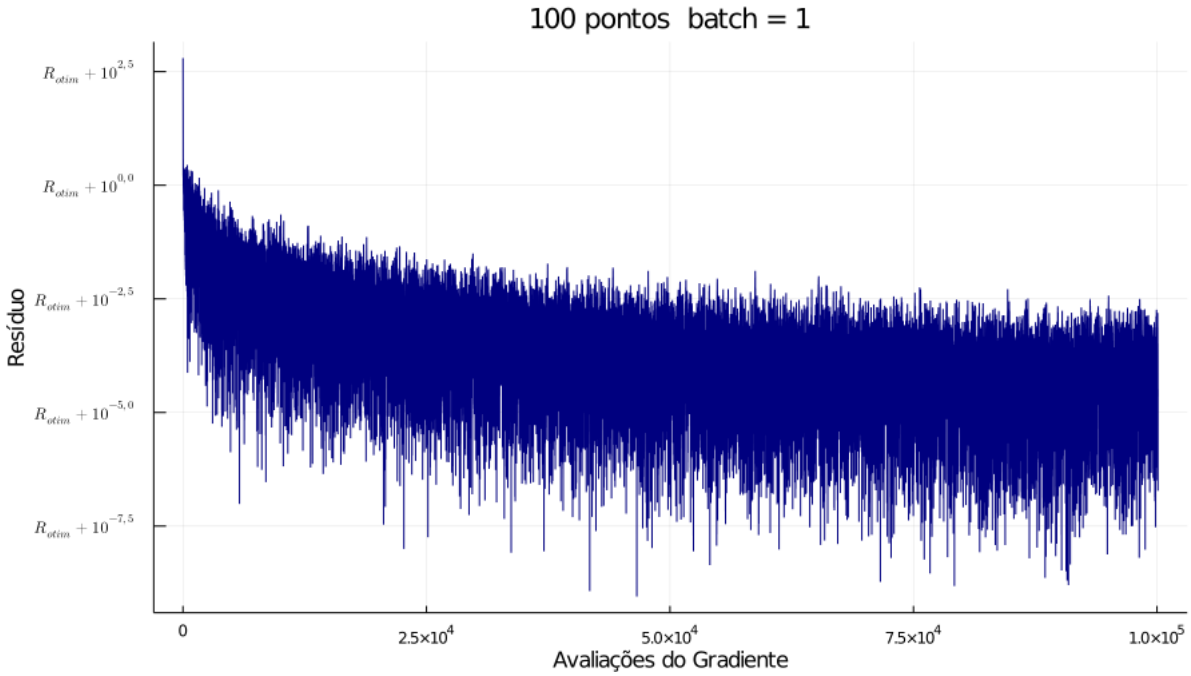


Figura 31: Decrescimento do modelo linear com $batch = 1$.

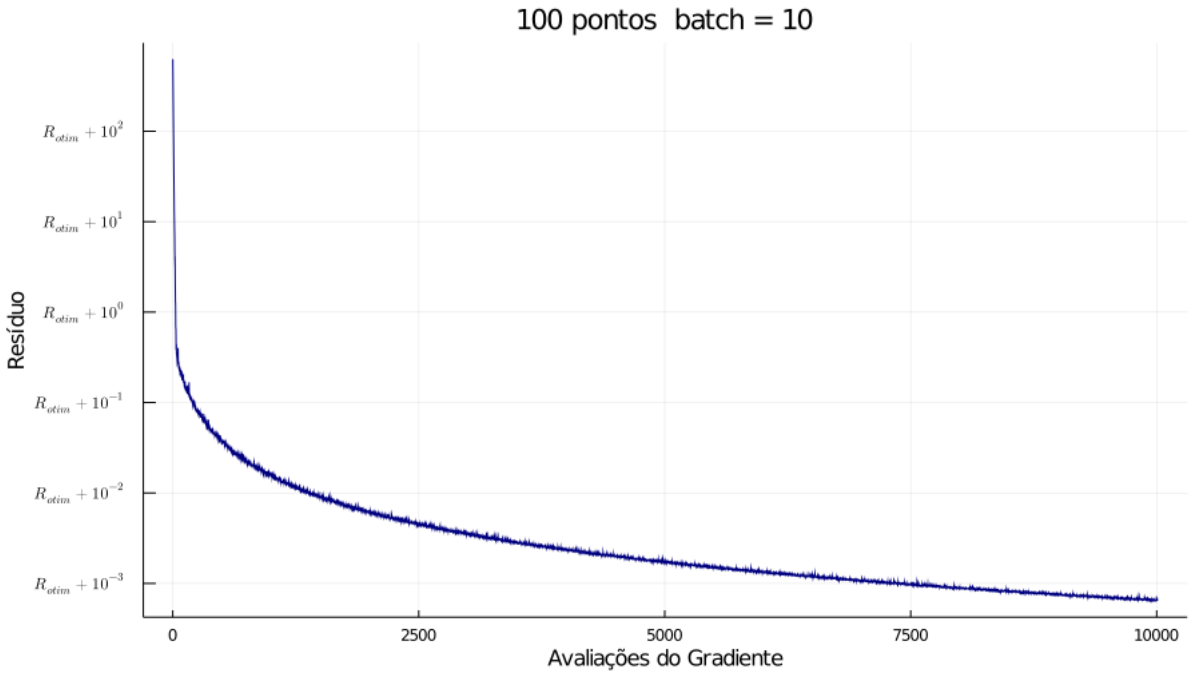


Figura 32: Decrescimento do modelo linear com $batch = 10$.

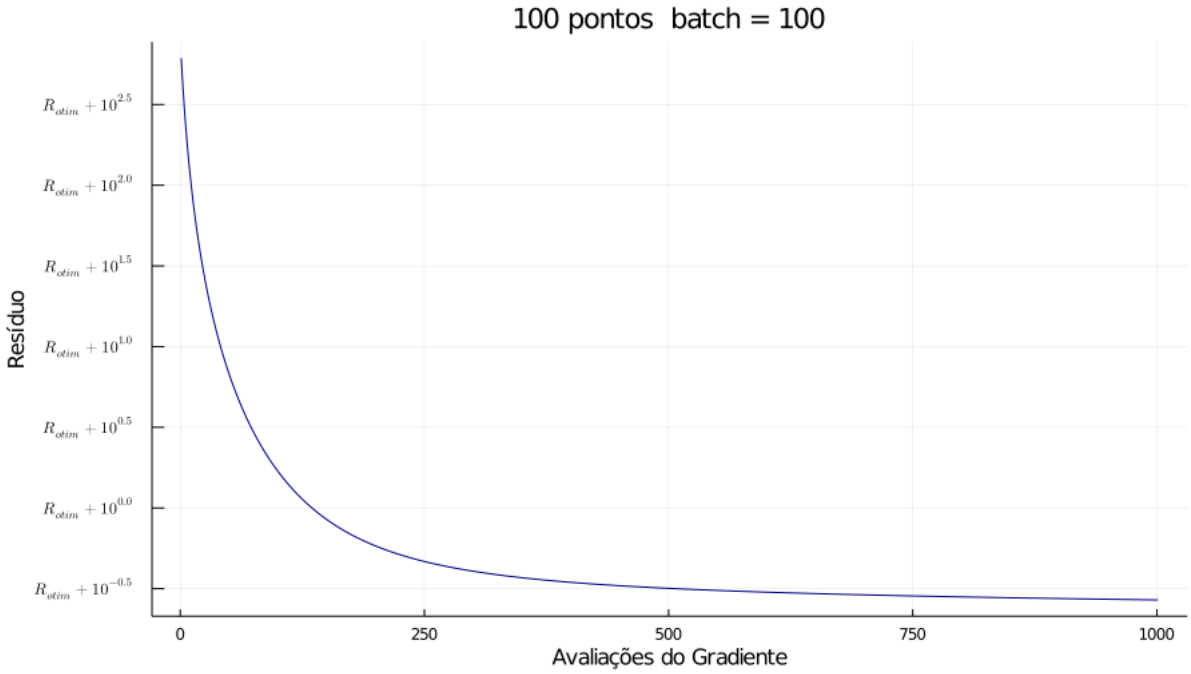


Figura 33: Decrescimento do modelo linear com $batch = 100$.

Como esperado, o Método do Gradiente Estocástico puro, $batch = 1$ tem uma grande variação residual devido ao seus passeios no trajeto de otimização. Já com o aumento do *batch*, tem-se uma drástica diminuição dessa variação.

Outro ponto importando que pode ser observado nas Figuras 31, 32 e 33 é a diminuição do número de Avaliações do Gradiente no eixo x, isso se dá ao fato de que do modo que foi escrito código do algoritmo, o valor de β é atualizado a cada vez que tem-se um *batch* completo, isso faz com que, por exemplo, com $batch = 1$ em um conjunto de 10 dados, a cada iteração o valor de β seja atualizado 10 vezes, mas considerando o $batch = 5$ no mesmo conjunto de dados, a cada iteração o valor de β seja atualizado apenas 2 vezes. Ou seja, para menores valores de *batch*, mais atualizações de β ocorrem, já com valores de *batch* maiores, menos atualizações de β ocorrem.

Aproveitando também estes gráficos de decrescimento, é possível entender a necessidade da utilização de um passo decrescente ao invés do constante. Considerando por exemplo a Figura 32, colocando $d = 0$, isso é, passo constante, fica mais clara a variância do modelo na Figura 34.

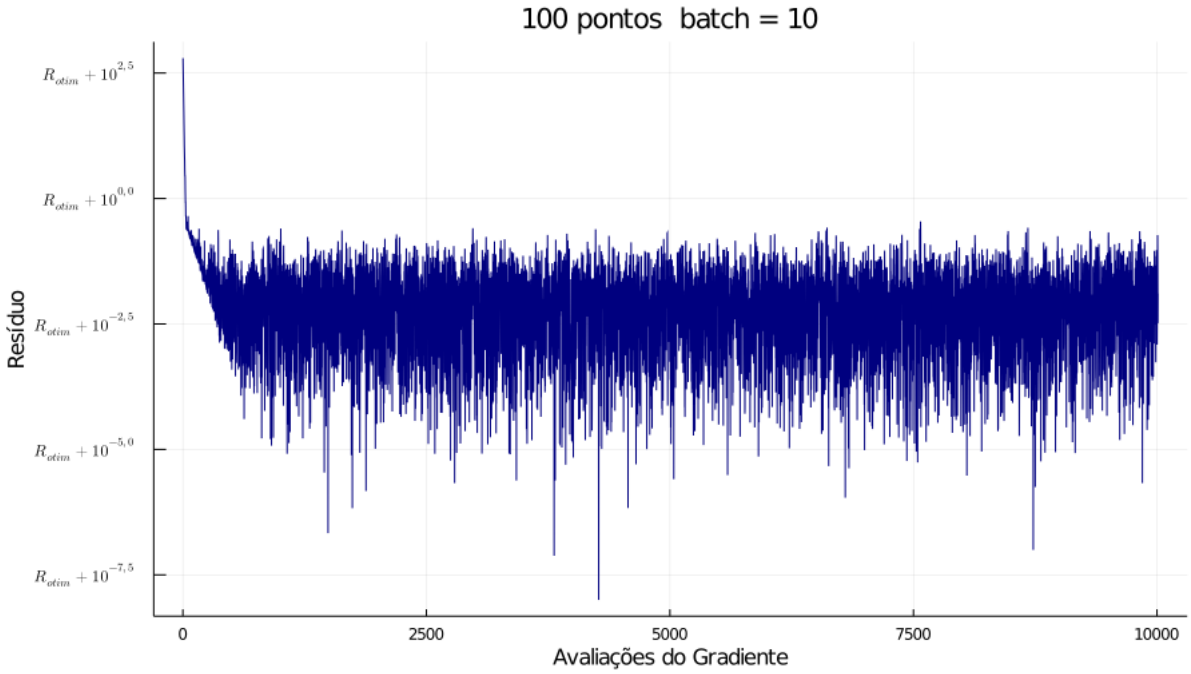


Figura 34: Decrescimento do modelo linear com *batch* = 10 e passo constante.

Variância esta que não ocorre no Gradiente Descendente com o passo constante 35.

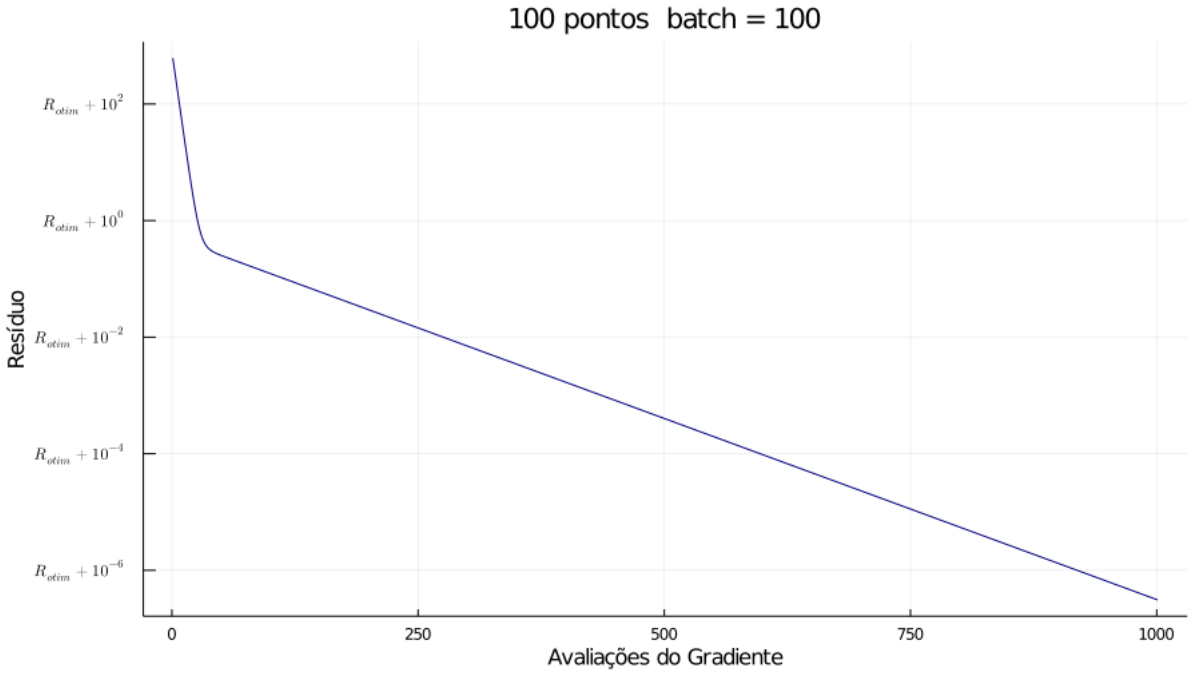


Figura 35: Decrescimento do modelo linear com Gradiente Descendente e passo constante.

Modelo Logístico

Seguindo então com o modelo logístico, a função *LogisticRegressionModel*. Como avaliação do resíduo gerado para cada tamanho de *batch* testado, foi utilizada a função

$$r(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

onde novamente, y consiste nos valores reais e $\hat{y}$ nos valores previstos pelo modelo.

O conjunto de pontos utilizados para a elaboração dos testes do modelo logístico consiste em um conjunto binário determinado pela seguinte função

$$y = \begin{cases} 1, & \text{se} \quad x_1^2 + x_2^2 > 0.7 + \varepsilon \\ 0, & \text{caso contrário} \end{cases}, \quad \varepsilon \sim N(0; 0{,}2)$$

Uma visualização de uma amostra de 100 pontos deste conjunto pode ser vista na Figura 36 a seguir na Figura 36.

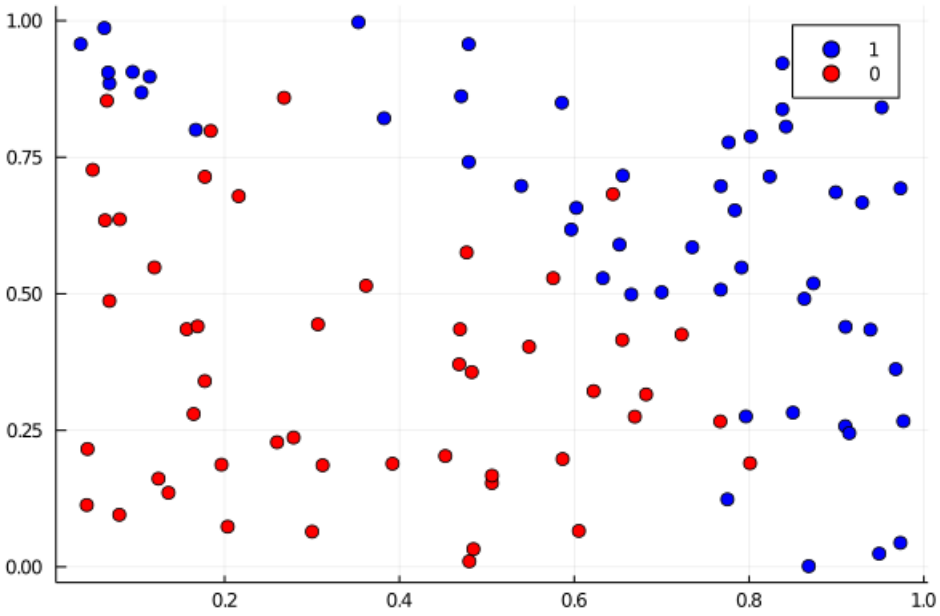


Figura 36: Amostra de 100 pontos do conjunto de pontos para o modelo logístico.

No modelo logístico, são necessárias algumas alterações dos *hiperparâmetros* para resultados com melhor qualidade neste conjunto de dados, estas são a tomada do coeficiente de regularização $\alpha = 10^{-6}$ e a taxa de aprendizado constante dada pelo passo *time-based* com $\gamma_0 = 3{,}0$ e $d = 0{,}5$. Com estas alterações aplicadas, novamente, considerando inicialmente o conjunto de 100 pontos, levou-se em consideração *batches* de tamanhos 1, 2, 5, 10, 20, 50 e 100, avaliando-se 1000 vezes o resíduo gerado por cada *batch* e com estes valores gerou-se os gráficos de dispersão a seguir.

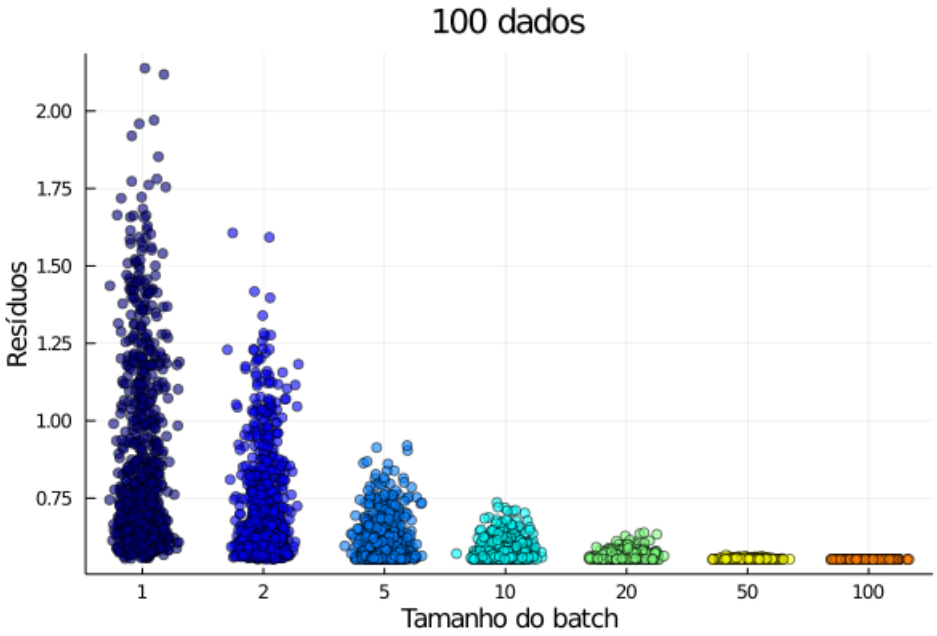


Figura 37: Dispersão do modelo logístico com 100 pontos.

Tal como no modelo linear, o conjunto de 200 pontos manteve o padrão de distribuições com os diferentes valores de *batches* na Figura 38.

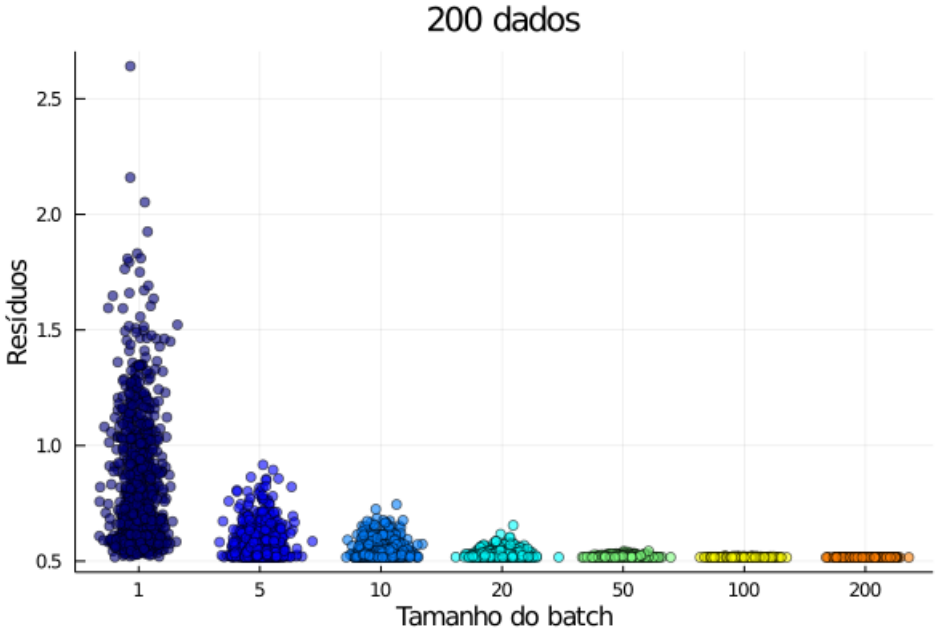


Figura 38: Dispersão do modelo logístico com 200 pontos.

Diferente do que foi realizado no modelo linear, consideramos aqui o gráfico do decrescimento do resíduo obtido de acordo com a quantidade de Avaliação do Gradiente. O resultado é visto nas Figuras 39, 40 e 41.

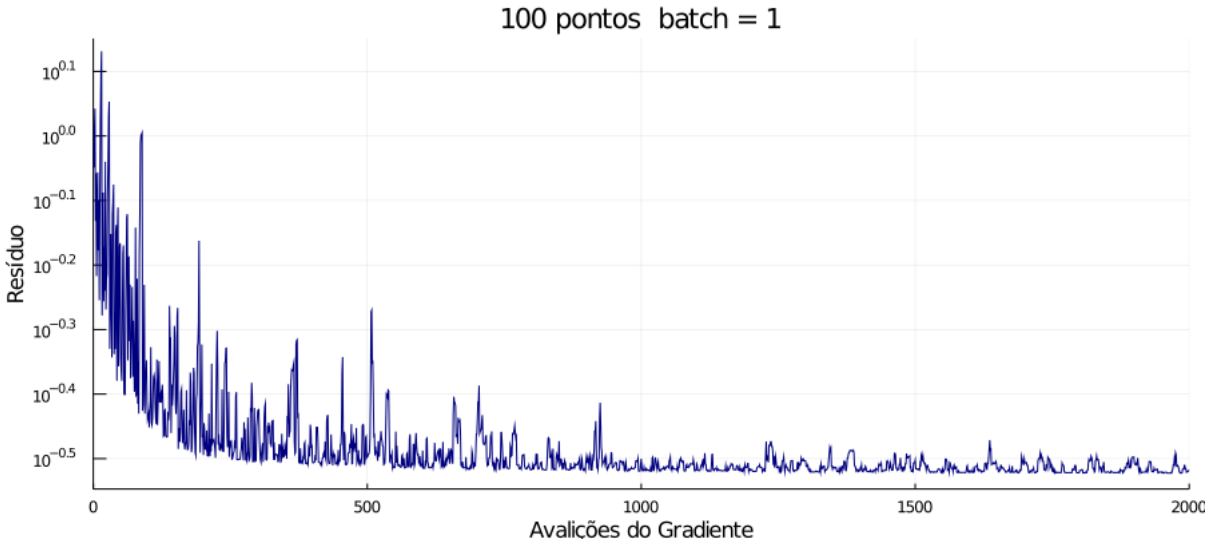


Figura 39: Convergência do *LogisticRegressionModel* com $batch = 1$.

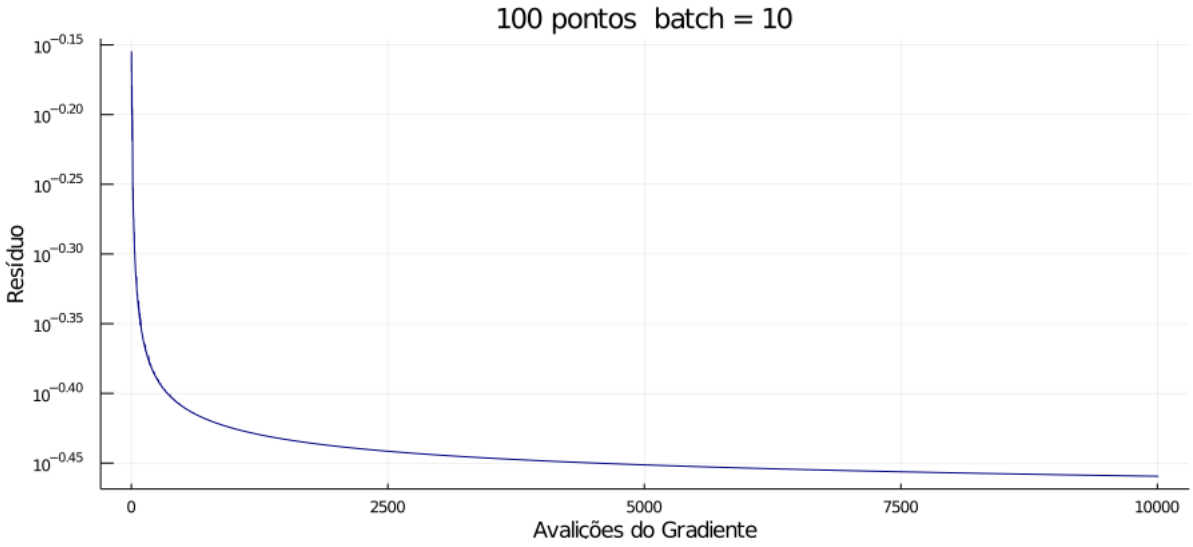


Figura 40: Convergência do *LogisticRegressionModel* com $batch = 10$.

Novamente, a variância do resíduo diminui conforme é aumentado o tamanho do *batch*. Neste caso ainda fica claro que considerando $batch = 10$ já se tem um equilíbrio entre a quantidade de Avaliações do Gradiente e o resíduo obtido, tendo em vista que o $batch = 100$, ou seja, o Gradiente Descendente, acaba sendo mais caro e neste caso determinando um resíduo maior, como pode ser visto na Figura 41 a seguir.

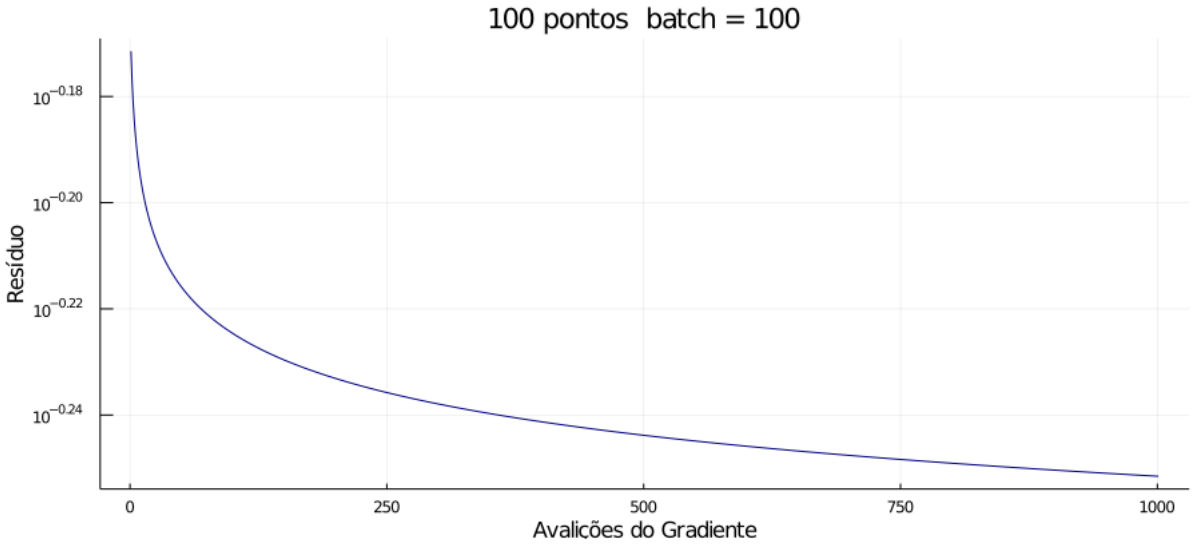


Figura 41: Convergência do *LogisticRegressionModel* com *batch* = 100.

Segundo Polyak [14], algo que pode contribuir na convergência do modelo é o que pode ser chamado de Gradiente Estocástico médio. A ideia deste consiste em, considerando nb o número de *batches* utilizados, dadas todas as atualizações de β no decorrer do algoritmo, definimos $\beta_{médio}$ como a média dos últimos nb-ésimos valores de β. Este coeficiente permite diminuir drasticamente a variância encontrada nas diferentes vezes que o algoritmo é executado. Essencialmente pode-se dizer que $\beta_{médio}$ torna o algoritmo com respostas mais constantes, principalmente considerando valores menores de *batches*.

Este efeito do $\beta_{médio}$ pode ser visualizado nas Figuras 42 e 43 a seguir. Considerando conjuntos com 100 pontos obtemos

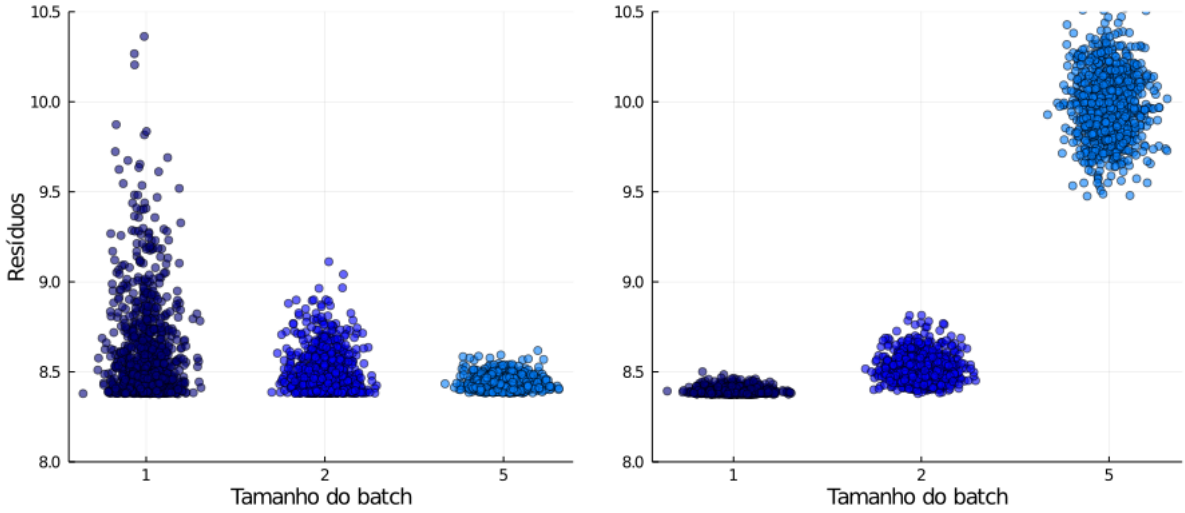


Figura 42: À esquerda os resultados do modelo linear com β comum e à direita os resultados com $\beta_{médio}$.

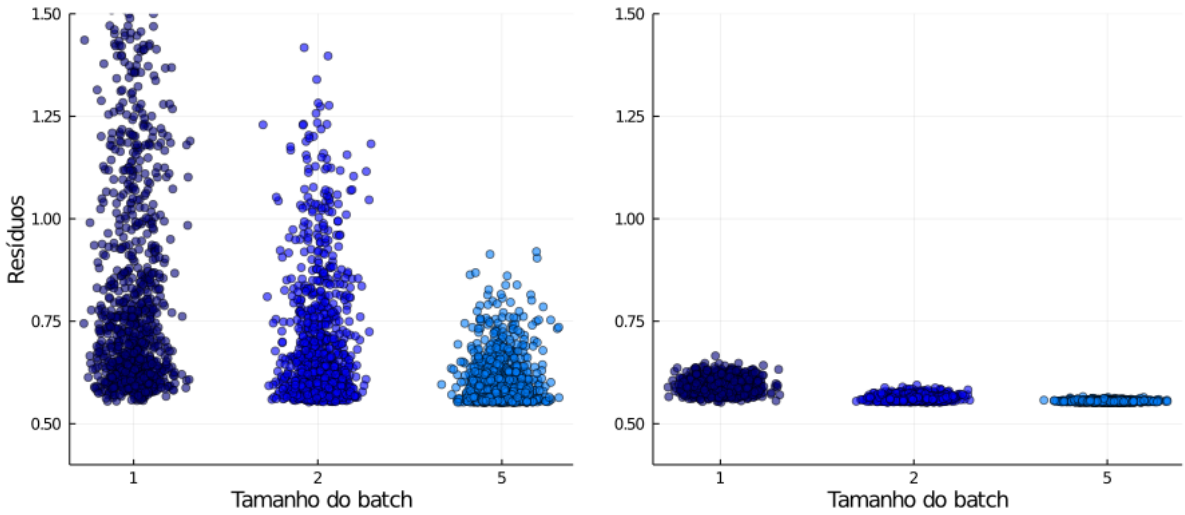


Figura 43: À esquerda os resultados do modelo logístico com β comum e à direita os resultados com $\beta_{médio}$.

Note que para o modelo logístico, $\beta_{médio}$ apresentou excelentes resultados, diminuindo a variância e até mesmo minimizando o resíduo quando considerado $batch = 1$. Já com o modelo linear, é possível observar que os resultados obtidos são bons porém, como pode ser visto no $batch = 5$, pode também ser de certa forma prejudicial ao modelo.

4.5 Implementação da Rede Neural com RegressionModels

Dada a estrutura do *RegressionModels*, como foi definida na Seção 4.2, esta recebe uma função $h(x, \beta)$ com algum modelo para ser ajustado a um conjunto de dados e uma função $\ell(y, \hat{y})$ para avaliar o erro entre o valor $\hat{y}$ previsto e o valor y original. Esta construção possibilita uma grande liberdade na escolha de modelos e funções de avaliação de erros, o que permite então utilizá-la na implementação de uma pequena Rede Neural.

O problema para o qual a rede foi implementada consiste no conjunto de dados *Wisconsin Breast Cancer Database* [5], referente ao diagnóstico de câncer de mama a partir de 699 amostras das 9 seguintes variáveis

- Espessura da aglomeração celular;
- Uniformidade do tamanho celular;
- Uniformidade da forma celular;
- Adesão marginal;
- Tamanho das células epiteliais individuais;
- Núcleos nus;

- Cromatina suave;
- Nucléolos normais;
- Mitoses raras.

Cada uma destas características é expressa por um valor inteiro entre 0 e 9 e devem ser utilizadas para determinar se a amostra pertence à classe benigna ou maligna.

Como é um problema de classificação binária, pode ser modelado como uma Regressão Logística. Sendo assim, a função para avaliação do erro no processo de treinamento da Rede Neural deve ser a função

$$\ell(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}).$$

Com a função para a avaliação do erro definida, resta agora a determinação da Rede Neural para ser utilizada na função $h(x, \beta)$.

Como foi apresentado na Seção 2.4, a maneira geral de definir uma Rede Neural de m camadas e K neurônios na camada interna é dada pela função

$$h_\beta(x) = g_{m+1}(B^{(m)} \cdot g_m(B^{(m-1)} \cdots g_1(B^{(1)}x + b_1) \cdots + b_{m-1}) + b_m),$$

onde $x \in \mathbb{R}^p$, $B \in \mathbb{R}^{K \times p}$ com $(B)_{kj} = \beta_{kj}$ com $j = 1, \ldots, p$ e $k = 1, \ldots, K$ e g_i são as funções de ativação.

Para a implementação neste trabalho, considerou-se uma Rede Neural com apenas uma camada escondida, sendo assim, a função fica definida como

$$h(x, \beta) = g_2(B^{(2)} \cdot g_1(B^{(1)}x + b_1) + b_2).$$

Agora, para a implementação desta função, o algoritmo deve ser escrito de uma maneira estratégica. Os parâmetros que são ajustados no treinamento da Rede Neural são as entradas das matrizes $B^{(1)}, B^{(2)}$ e os valores em b_1, b_2, sendo assim todos estes devem estar ligados aos valores do vetor β, que é o vetor atualizado pelo algoritmo do Gradiente Estocástico. Tendo isso em vista, o vetor β deve ter o tamanho inicial dado por $pK + 2K + 1$, onde p é o número de variáveis do problema e K o número de neurônios da camada escondida.

Com o vetor β sendo iniciado com esta quantidade de valores randômicos, é possível seccioná-lo da seguinte forma:

- $B^{(1)}$ contém valores dos índices $1, \ldots, Kp$;
- b_1 contém os valores dos índices $Kp + 1, \ldots, Kp + K$;
- $B^{(2)}$ contém os valores dos índices $Kp + K + 1, \ldots, Kp + 2K$;

- b_2 contém o valor do índice $Kp + 2K + 1$.

Assim, o vetor β é atualizado pelo algoritmo do Gradiente Estocástico e reorganizado para os cálculos na Rede Neural.

Para as funções de ativação de cada camada, sendo um problema de classificação binária, utilizou-se

$$g_1(z) = g_2(z) = \sigma(z) = \frac{1}{1 + e^{-z}},$$

ou seja, a própria função sigmoide.

Para a avaliação dos acertos utilizou-se a matriz de confusão, dada por

		Valor Previsto	
		Negativo	Positivo
Valor Esperado	Negativo	T_n	F_p
	Positivo	F_n	T_p

onde

- T_n (Verdadeiro Negativo) - Rótulos 0 corretos;
- T_p (Verdadeiro Positivo) - Rótulos 1 corretos;
- F_n (Falso Negativo) - Rótulos 0 errados;
- F_p (Falso Positivo) - Rótulos 1 errados.

E também o cálculo da acurácia dos acertos, dado por

$$acc = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}.$$

Para as avaliações considerou-se apenas o Método do Gradiente Estocástico puro. Inicialmente utilizando 300 dados como treinamento e somente $K = 2$ neurônios na camada escondida, a matriz obtida é dada por

		Valor Previsto	
		Negativo	Positivo
Valor Esperado	Negativo	104	0
	Positivo	22	273

possibilitando uma acurácia de 0,94 em aproximadamente 9,41 segundos de execução. Tendo em vista a simplicidade do modelo e a pouca quantidade de dados utilizados no teste, os resultados são bastante satisfatórios. Como comparação foi executado o modelo *LogisticRegressionModel* também com Gradiente Estocástico puro e os resultados foram

		Valor Previsto	
		Negativo	Positivo
Valor Esperado	Negativo	103	1
	Positivo	58	237

determinando uma acurácia de 0,85 em aproximadamente 2,30 segundos de execução. Sendo mais rápido, porém ficando para trás na acurácia.

Mantendo esta quantidade de dados e aumentando a quantidade de Neurônios da camada interna da Rede Neural para $K = 4$, tem-se um aumento da acurácia para 0,97 porém o tempo de execução passa a ser de aproximadamente 23,25 segundos.

O aumento da quantidade de dados de treinamento acaba melhorando para os dois modelos, sendo a Rede Neural ainda com $K = 2$, tem-se

		Valor Previsto	
		Negativo	Positivo
Valor Esperado	Negativo	44	0
	Positivo	2	153

com uma acurácia de 0,98 em 15,00 segundos de execução. E com o modelo logístico

		Valor Previsto	
		Negativo	Positivo
Valor Esperado	Negativo	44	0
	Positivo	11	144

com a acurácia de 0,94 e 3,61 segundos de execução.

Novamente, aumentando o número de neurônios da camada interna para $K = 4$, tem-se um aumento da acurácia para 0,99, porém o tempo também tem um aumento para 40,28 segundos. A Rede Neural implementada, por mais simples que seja, apresenta resultados interessantes na aplicação porém, acaba ficando para trás quando o tempo de execução

é considerado, o que é um fator bastante relevante em algoritmos de Aprendizagem de Máquina.

4.6 Comparação com o pacote *scikit-learn*

Finalmente, após a finalização do estudos acerca das influências dos diferentes tipos de passos e dos resultados obtidos de acordo com os diferentes tamanhos de *batch*. O objetivo agora é fazer uma comparação direta com o pacote *scikit-learn* [13] em linguagem Python.

O *scikit-Learn* é de uma biblioteca em Python gratuita e de código livre focada em Aprendizagem de Máquina. Atualmente, é uma das bibliotecas referência neste contexto. Tendo isso em vista, a comparação aqui desenvolvida surge para analisar, de certa forma, a qualidade da implementação proposta neste trabalho.

Para a realização desta comparação, buscou-se um conjunto de dados de tamanho considerável para que seja possível colocar os algoritmos de ambas as linguagens à prova. Como conjunto de dados, utilizando a plataforma *Kaggle* [9] de Aprendizagem de Máquina, escolheu-se "HR Analytics Case Study", um conjunto de dados que contém 15000 linhas e 10 colunas. Este conjunto de dados contém informações sobre o seguinte problema: Todo ano, aproximadamente 15% dos funcionário deixam uma determinada empresa, esta quer descobrir quais motivos são os mais presentes nesta decisão dos funcionários que saem. Para isso, as seguintes informações são fornecidas:

- Satisfação do funcionário (Entre 0 e 1);
- Última avaliação do funcionário (Entre 0 e 1);
- Número de projetos que se envolveu;
- Média mensal de horas trabalhadas;
- Anos trabalhados na empresa;
- Teve algum acidente de trabalho (0 ou 1);
- Teve alguma promoção nos últimos 5 anos (0 ou 1);
- Salário (alto, médio ou baixo);
- Departamento;
- Deixou a empresa (0 ou 1).

Ou seja, dadas as nove informações iniciais, deve-se prever se o funcionário deixará ou não a empresa. Um problema que pode ser modelado como uma Regressão Logística.

Considerando 11000 linhas para treino e 4000 para teste, como métricas para a análise da qualidade das soluções foram utilizadas a matriz de confusão.

Aplicando o Método do Gradiente Estocástico puro na resolução, inicialmente em Julia, foram utilizados os parâmetros: $\gamma_0 = 10^{-4}$, passo *optimal* e no máximo 100 iterações. A matriz de confusão obtida desta forma é dada por

		Valor Previsto	
		Negativo	Positivo
Valor Esperado	Negativo	1574	854
	Positivo	978	593

que resultou em uma acurácia de 0,54 e 4,82 segundos de tempo.

Por outro lado, resolvendo o mesmo problema com o pacote *scikit-learn* em Python, utilizou-se o método chamado *SGDClassifier* (Sthocastic Gradient Descent Classifier) e o único parâmetro alterado foi a função erro, agora mudada para a mesma função em (4), utilizada na implementação do modelo em Julia. Neste caso, considerando também no máximo 1000 iterações e o passo *optimal*, o resultado obtido foi dado por

		Valor Previsto	
		Negativo	Positivo
Valor Esperado	Negativo	1641	786
	Positivo	854	717

com uma acurácia de 0,58 utilizando apenas 0,33 segundos.

Retomando com o pacote em Julia, considerando o passo *exponential* com $\gamma_0 = 19$, foi possível obter um resultado ainda melhor, tanto em velocidade quanto em acurácia, obtendo-se então

		Valor Previsto	
		Negativo	Positivo
Valor Esperado	Negativo	1478	950
	Positivo	534	1037

com uma acurácia de 0,62 em apenas 2,01 segundos.

Um resultado semelhante a este foi obtido em Python considerando o passo *invscaling* também com $\gamma_0 = 19$, obtendo-se

		Valor Previsto	
		Negativo	Positivo
Valor Esperado	Negativo	1200	228
	Positivo	898	672

resultando em uma acurácia também de 0,62 porém em 0,13 segundos.

Tendo em vista que os pontos tomados no Método do Gradiente Estocástico têm o fator randômico incluso, isto é, os pontos tomados para a avaliação do erro são sorteados (Algoritmo 2), cada vez que os métodos são iniciados, pode obter-se resultados diferentes ao término da execução. Mesmo com este fator, os algoritmos de ambas as bibliotecas se mostraram bastante semelhantes no sentido de acurácia, como é possível observar nas matrizes de confusão dos mesmos.

O problema da comparação das bibliotecas acaba ficando principalmente no quesito tempo. Com a biblioteca em Python atingindo frações de segundos para a execução, o método implementado em Julia acaba ficando para trás. Porém, vale lembrar que a biblioteca em Python já vem sendo desenvolvida e otimizada há alguns anos, enquanto o método aqui implementado tem apenas alguns meses de existência. O que mostra que, com a futura otimização desta biblioteca em Julia, é possível se obter resultados bastante promissores.

Uma maneira de diminuir o tempo de execução do algoritmo em Julia é a utilização do *mini-bacth*. Sendo assim, utilizando $\gamma_0 = 10^{-4}$, máximo de 100 iterações e $batch = 10$, obtém-se

		Valor Previsto	
		Negativo	Positivo
Valor Esperado	Negativo	2103	325
	Positivo	1434	137

com acurácia de 0,56, em 0,76 segundos. Finalmente atingindo as frações de segundos.

Para um resultado ainda mais rápido mantendo a qualidade, com $batch = 10$, foi utilizado $\gamma_0 = 15$, máximo de 100 iterações e o passo *exponential*, obtendo-se

		Valor Previsto	
		Negativo	Positivo
Valor Esperado	Negativo	1489	939
	Positivo	738	833

com acurácia de 0,58 e 0,24 segundos de execução.

Aumentando o tamanho do *batch* para 100, ainda foi possível obter resultados satisfatórios. Com $\gamma_0 = 10^{-2}$ e passo *optimal* foi possível obter a matriz

		Valor Previsto	
		Negativo	Positivo
Valor	Negativo	1387	1041
Esperado	Positivo	584	987

que resultou numa acurácia de 0,59 em também 0,24 segundos.

A partir deste tamanho de *batch*, o algoritmo consegue algumas vezes executar com mais velocidade, porém a acurácia acaba sendo perdida pois o modelo passa a realizar *overfitting*.

5 Considerações finais

O objetivo deste trabalho foi estudar e implementar os algoritmos de Gradiente Descendente e Gradiente Estocástico com *mini-batch*. A implementação destes algoritmos da Aprendizagem de Máquina tinha como foco a resolução de problemas de Regressão Linear e Logística e a realização de um comparativo com uma das bibliotecas deste contexto mais conhecidas da atualidade, o pacote *scikit-learn* em linguagem Python.

A implementação do algoritmo foi realizada com sucesso, atingindo as expectativas existentes anteriores ao projeto e trazendo bons resultados no momento de sua aplicação em conjuntos de dados diversos, permitindo até mesmo a implementação de uma pequena Rede Neural a partir da estrutura do *RegressionModels*.

As maiores dificuldades encontradas no decorrer do projeto foram os ajustes dos *hiperparâmetros*. Estes, quando mal definidos, podem fazer com que o desempenho do algoritmo seja bastante inferior ao possível. Sendo assim, para cada teste realizado em diferentes conjuntos de dados era necessária a dedicação de tempo para se obter o potencial do método.

O próximo passo no desenvolvimento do pacote em linguagem Julia trata-se da otimização do algoritmo, visando torná-lo mais eficiente e principalmente mais rápido.

Referências

[1] BEZANSON, J., EDELMAN, A., KARPINSKI, S. e SHAH, B. **Julia: A Fresh Approach to Numerical Computing**. *SIAM Review 59*, 1 (Jan. 2017), 65 - 98.

[2] BOTTOU, L. **Stochastic Gradient Tricks**. In *Neural Networks, Tricks of the Trade, Reloaded*, G. Montavon, G. B. Orr, e K-R. Müller, Eds., Lecture Notes in Computer Science (LNCS 7700). Springer, 2012, pp. 430-445.

[3] BOTTOU, L., CURTIS, F. E. e NOCEDAL, J. **Optimization Methods for Large-Scale Machine Learning**. *SIAM Review 60*, 2 (Jan. 2018), 223-311.

[4] BRASIL, C. **Painel Coronavirus**. Disponível em: `https://covid.saude.gov.br`. Acesso em 02 de dezembro de 2020.

[5] DUA, D., CASEY, G. **UCI Machine Learning Repository**. Disponível em: `http://archive.ics.uci.edu/ml`. Acesso em 02 de dezembro de 2020.

[6] FRIEDLANDER, A. **Elementos de Programação Não-Linear**. Disponível em: `https://www.ime.unicamp.br/~friedlan/livro.htm`. Acesso em 02 de dezembro de 2020.

[7] GARETH, J., WITTEN, D., HASTIE, T. e TIBSHIRANI, R. **An Introduction to Statistical Learning**. Springer New York, 2013.

[8] HIGHAM, C. F., HIGHAM, D. J. **Deep Learning: An Introduction for Applied Mathematicians**. *SIAM Review 61*, 3 (Jan. 2019), 860-891.

[9] KAGGLE. **HR Analytics Case Study**. Disponível em: `https://www.kaggle.com/vjchoudhary7/hr-analytics-case-study`. Acesso em 02 de dezembro de 2020.

[10] MEYER, P. L. **Probabilidade: Aplicações à Estatistica**, LTC, 1987.

[11] NESTEROV, Y. **Lectures on Convex Optimizations**. Springer International Publishing, 2018.

[12] PATTERSON, J, GIBSON, A. **Deep Learning: A Practioner's Approach**, Packt Publishing Ltd., 2017.

[13] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M. e DUCHESNAY, E. **Scikit-learn: Machine Learning in Python**. *Journal of Machine Learning Research 12* (2011), 2825-2830.

[14] POLYAK, B. T., JUDTSKY, A. B. **Acceleration of Stochastic Approximmation by Averaging**. *SIAM Journal on Control and Optimization 30*, 4 (Julho 1992), 838-855.

[15] RONCHI, C. H. V. **Estudo Matemático do Reconhecimento de Caracteres**. *Trabalho de Conclusão de Curso (bacharelado em Matemática)*, (2017).

[16] SHWARTZ, S. S. e DAVID, S. B. **Understanding Machine Learning: From Theory to Algorithms**. Cambridge University Press, Estados Unidos da América, Nova Iorque, 2014.

[17] ZHOU, X. **On the Fenchel Duality between Strong convexity and Lipschitz Continuous Gradient**, 2018.