MAC0215

Fundamentos de Redes Neurais

Aluno Heitor Barroso Cavalcante - 12566101
Orientadora Nina S. T. Hirata

16 de Dezembro de 2022

Conteúdo

1 Introdução

Esse texto tem como objetivo introduzir noções basilares para a compreensão de conceitos fundamentais do campo de estudo de aprendizado de máquina, mais especificamente de redes neurais. Para fazer isso, primeiro será apresentada a ideia geral do aprendizado baseado em dados, que trespassa todos os temas abordados durante o texto. Depois disso, estudaremos dois algoritmos importantes para o entendimento de redes neurais: Regressão Linear e Logística; e, por fim, tendo construído o arcabouço necessário, abordaremos o tópico de redes neurais.

2 Aprendizado Baseado em Dados

Entre as incontáveis situações do cotidiano em que são aplicadas técnicas de aprendizado de máquina, podemos ressaltar algumas aplicações instigantes como algoritmos de recomendação de filmes, reconhecimento facial e previsões financeiras. Atravessando os mais diversos temas e áreas do conhecimento, essas técnicas são utilizadas bem sucedidamente pois os problemas que elas se propõe a resolver têm, em comum, um processo de solução baseado em exemplos.

Em outras palavras, imagine que você, com duas fotografias em mãos — uma de um cachorro e a outra de um gato — pedisse a um amigo para diferenciar os dois animais. Certamente o amigo conseguiria distinguir o cachorro do gato com certa facilidade. Contudo, se a tarefa fosse descrever objetivamente as características que diferenciam um cachorro de um gato, com certeza esse processo se tornaria mais complicado.

Isso acontece pois, ao longo de nossas vidas, aprendemos a diferenciar cachorros de gatos por meio de exemplos. Já vimos tantos cães e tantos gatos que nossos cérebros se tornam treinados, com base nessas experiências passadas, para distinguir ambos os animais.

Continuando essa linha de raciocínio, a solução dos problemas citados também passa por esse caminho. Mesmo que difíceis de identificar, com certeza há características comuns nos filmes que um telespectador gosta de assistir, assim como certamente há padrões nos dados que registram o comportamento de um ativo financeiro.

Desse modo, o ponto central dessas técnicas de aprendizado de máquina

— que serão introduzidas e explicadas ao longo desse texto — é, por meio de exemplos, "treinar" modelos para que sejam capazes de identificar esses padrões. E, então, quando nos depararmos com situações novas (como decidir se gostaremos ou não de um filme inédito), poderemos usar o modelo treinado para balizar decisões (que terão sido baseadas em experiências passadas).

2.1 Abstração do Aprendizado

Se pensarmos bem, a preferência de uma pessoa por filmes pode ser interpretada como uma função, no sentido matemático da palavra. Isto é, o fato desse indivíduo gostar ou não de um "filme x" obedece uma função f(filme x). No decorrer do texto, nos referiremos à essa função f como "função alvo". Agora, de que modo um filme poderia ser transformado em parâmetro dessa função? Há diversas maneiras de responder a essa questão.

- Poderíamos representar a obra cinematográfica como sua nota em sites de crítica especializada.
- Talvez representar cada filme de acordo com uma medida arbitrária da quantidade de ação, comédia e drama que o constituem.
- Ou, simplesmente, reduzir o filme ao seus dois gêneros predominantes.

Independente disso, considerando os filmes que esse sujeito já assistiu e quais deles ele gostou ou não, podemos representar esses exemplos na forma dos seguintes pares de dados:

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$$

Sendo $\mathbf{x}_i,\ i = 1, \ldots, N$ cada filme e $y_i,\ i = 1, \ldots, N$ uma variável indicando se a pessoa gostou ou não do filme $\mathbf{x}_i$. Perceba que estamos representando cada filme $\mathbf{x}_i$ em negrito pois, geralmente, um vídeo será representado como um vetor. Tal vetor pode conter a quantidade de ação, comédia e drama que formam o filme ou possuir os dois gêneros mais pronunciados ao longo da obra; o conteúdo do vetor $\mathbf{x}_i$ depende da maneira com a qual representaremos o filme para ser usado como entrada da função.

Todos os problemas que resolvemos utilizando técnicas de aprendizado de máquina sem importar se os dados de entrada são imagens, filmes, gravações de áudio, planilhas financeiras, palavras ou números, seguirão esse padrão de representação em pares de dados.

Na prática, se um problema requer Machine Learning, não conseguimos obter uma solução analítica que nos forneça a lei da função f. Portanto, usaremos essas técnicas de aprendizado de máquina para encontrarmos uma função $h(\mathbf{x}_i) = \hat{y}_i$ que tenha comportamento o mais semelhante possível em relação à f. Isto é, tal que $\hat{y}_i$ seja o mais próximo possível de y_i, para $i = 1, 2, \ldots, N$.

2.1.1 Modelo Linear

Agora que já temos definido como seria a entrada dessa função ($\mathbf{x}_i$) e sua saída ($\hat{y}_i$), precisamos pensar como seria a função propriamente dita. Primeiramente, podemos elencar dois tipos principais de problemas: os de classificação e os de regressão.

2.1.1.1 Classificação

Os problemas de classificação podem ser interpretados como problemas em que, dado uma entrada $\mathbf{x}_i$, o valor que iremos prever (y_i) — perceba que iremos prever um valor de y_i, mas o valor que iremos obter efetivamente é $\hat{y}_i$ — simboliza uma variável discreta. Repare que no caso de prever se gostaremos ou não de um filme, a variável y_i é binária, e só poderia assumir, por exemplo, os valores 1 ou 0. Em outra situação, de diferenciar cachorros de gatos, poderíamos utilizar $y_i = 0$ para representar gatos e $y_i = 1$ cachorros.

De qualquer forma, quando estamos tratando de problemas de classificação, alguns deles podem ser caracterizados como linearmente separáveis. Isso quer dizer que, a partir de todas as entradas $\mathbf{x}_i$ já observadas — que utilizaremos para treinar a máquina — há uma estrutura linear (hiperplano) que separa os dados em suas devidas classes. Conseguiremos visualizar essa característica através da implementação efetuada na seção sobre Regressão Logística, em que decidiremos se um cliente i comprou ($y_i = 1$) ou não ($y_i = 0$) um determinado produto.

2.1.1.2 Regressão

Por outro lado, nos problemas de regressão, apesar de termos a mesma entrada $\mathbf{x}_i$, y_i será uma variável contínua. No lugar de atribuir à entrada alguma classe, nós obtemos um valor contínuo. Por exemplo, dado o histórico escolar, estimar o salário futuro de estudantes universitários ou um banco decidir a quantidade ideal de crédito que se deve

fornecer para um dado cliente. Em vista disso, há situações em que o valor da variável y_i varia de maneira consideravelmente linear de acordo com a entrada $\mathbf{x}_i$.

Após considerarmos os problemas que podem ser resolvidos de maneira direta usando modelos lineares, é natural surgir indagações sobre quais ferramentas utilizaríamos caso esse caráter de certa linearidade nos dados não esteja presente. Nesse sentido, para resolver questões que não apresentam tais características de linearidade, o procedimento aplicado é o de combinar, diversas abordagens lineares — injetando não linearidade entre essas abordagens, de uma maneira que será explicada na seção sobre Redes Neurais — e, dessa forma, conseguir resolver o problema. Estudaremos mais esse tópico na seção sobre Redes Neurais.

Assim, como todas as técnicas de aprendizado de máquina cobertas nesse texto se apoiam em pilares de linearidade, definiremos o Modelo Linear, que estará fortemente presente de agora em diante.

Desse modo, unindo as ideias apresentadas até aqui, a ideia central dos algoritmos abordados ao longo das próximas seções é representar os problemas que desejamos solucionar como funções que possuem uma essência, de certa forma, linear. Contudo, apesar de termos conhecimento sobre o formato dessa função, não a conhecemos de fato.

Nesse sentido, para adquirir uma melhor noção sobre essa função h que queremos construir para aproximar f, é válido pensar na representação de dados como exemplos:

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$$

Examinando mais a fundo um vetor $\mathbf{x}_i$ que esteja dentro desse conjunto de exemplos, temos o que segue:

$$\mathbf{x}_i = \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,d} \end{bmatrix}$$

Os elementos $x_k,\ k = 1, \ldots, d$ desse vetor são chamados de características (*features*) da entrada e d é a dimensão dos vetores de entrada $\mathbf{x}$. Voltando ao exemplo dos filmes, e considerando $d = 3$ essas features podem ser tais que:

- $x_{i,1}$ representa a quantidade de ação no filme i
- $x_{i,2}$ representa a quantidade de aventura no filme i
- $x_{i,3}$ representa a quantidade de drama no filme i

Agora, sabendo que uma função linear é da forma:

$$h(x_1, x_2, \ldots, x_d) = a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_d x_d$$

Vamos considerar um vetor $\mathbf{w}$ (o qual chamaremos de vetor de pesos) tal que:

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$$

Dessa forma, queremos relacionar os pesos $w_k,\ k = 0, \ldots, d$ com as features $x_{i,k},\ k = 1, \ldots, d$ de cada exemplo $\mathbf{x}_i$ de maneira linear tentando obter uma expressão que aproxime o comportamento da função f que leva $\mathbf{x}_i$ a y_i, com $i = 1, \ldots, N$.

A fim de realizar tal tarefa, vamos relacionar $\mathbf{x}_i$ e $\mathbf{w}$ da seguinte maneira:

$$h(\mathbf{x}_i) = w_0 + w_1 x_{i,1} + w_2 x_{i,2} + \ldots + w_d x_{i,d}$$

Para facilitar a notação, podemos inserir o elemento $x_{i,0} = 1$ em cada vetor de exemplo $\mathbf{x}_i$. Denotaremos $\mathbf{x}_i$ adicionado desse elemento por $\tilde{\mathbf{x}}_i$:

$$h(\mathbf{x}_i) = w_0 + w_1 x_{i,1} + w_2 x_{i,2} + \ldots + w_d x_{i,d} = \mathbf{w}^T \tilde{\mathbf{x}}_i$$

Esse é o modelo linear. Por fim, o processo por trás da abstração de aprendizado envolvida em Machine Learning consiste em utilizar os exemplos de dados $(\mathbf{x}_i, y_i)$ para produzirmos h que aproxime f. Isso será feito encontrando o vetor de pesos $\mathbf{w}$ que melhor faça esse papel. A maneira de encontrar esse valor ótimo para $\mathbf{w}$ é fundamentada pela chamada "Função de Perda", que explicaremos a seguir.

2.2 Função de Perda

Função de Perda é a função que nos dirá o quão ruim, para aproximar f, h está. Essa função pode variar de caso a caso, mas a ideia central desse conceito é, de alguma forma, comparar os valores y_i — oriundos do nosso

conjunto de exemplos — com os resultados $h(\mathbf{x}_i) = \hat{y}_i$ em uma expressão que esteja em função do vetor de pesos $\mathbf{w}$. O resultado dessa comparação deve corresponder à quão diferente a função alvo (desconhecida) é da função h, que obtemos. Assim, para encontrar a função h que mais se assemelha da função alvo, deveremos buscar por valores mínimos da função de perda.

2.3 Otimização

Desse modo, a questão central das técnicas de aprendizado de máquina se reduz, de certa forma, à minimizar uma função de perda. Objetivando encontrar h que mais se aproxima da função alvo (f), podemos encarar tal questão como um problema de otimização. Apesar de haver diversas maneiras de se otimizar (nesse caso, minimizar) funções, é muito comum utilizarmos estratégias numéricas e, mais especificamente, o método do gradiente descendente.

2.3.1 Gradiente Descente

A ideia desse algoritmo de otimização é, dado uma função côncava, para encontrar o ponto de mínimo, basta "caminhar" sempre na direção de variação mais negativa da função a partir do ponto atual. Como será explicado, essa direção é a direção do vetor oposto ao gradiente da função no ponto. Com isso, teremos o seguinte algoritmo de atualização do vetor de pesos $\mathbf{w}$:

$$\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) - \eta \frac{\nabla l(\mathbf{w})}{\|\nabla l(\mathbf{w})\|}$$

Onde η é a chamada "taxa de aprendizado".

2.3.1.1 Máxima Variação

Consideremos o vetor unitário $\vec{v}$. E uma função f qualquer. A derivada direcional no sentido desse vetor é:

$$\frac{\partial}{\partial \vec{v}} f(x) = \nabla f(x) \cdot \vec{v} = \|\nabla f(x)\| \|\vec{v}\| \cos(\alpha) = \|\nabla f(x)\| \cos(\alpha)$$

Perceba que essa expressão assume valor máximo se e somente se $\cos(\alpha) = 1$. Ou seja, se o vetor $\vec{v}$ tem mesma direção que o vetor $\nabla f(x)$. Pela mesma justificativa, podemos concluir que a variação mais negativa ocorre quando $\vec{v} = -\frac{\nabla f(x)}{\|\nabla f(x)\|}$.

Portanto, se desejarmos maximizar (Gradiente Ascendente) ou minimizar (Gradiente Descendente) uma função, podemos usar esse método.

É válido comentar que, apesar da vantagem evidente de se usar tal método em funções côncavas (convergência ao mínimo global da função), também o utilizamos para minimizar funções de perda que não são necessariamente côncavas. Esse é o caso de Redes Neurais, por exemplo. Nesses casos, como a função não é côncava, existe a possibilidade de, no processo de "descida do gradiente", acabarmos ficando presos em mínimos locais. Contudo, apesar disso, esse método de otimização é amplamente utilizado em Machine Learning e, inclusive, existem algumas práticas – ou parâmetros – que são utilizadas para tentar diminuir as ocorrências desses "aprisionamentos" do gradiente em mínimos locais.

3 Regressão Linear

Primeiramente, de maneira simples, utilizamos Regressão Linear para encontrar uma relação linear entre dados de entrada e saída.

Exemplo 3.1 *Podemos pensar na relação de altura e peso nos seres humanos. Sendo a altura a variável independente e o peso a variável dependente. Disso, temos o conjunto de dados:*

$$(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)$$

Sendo $x_i = altura\ da\ pessoa_i \quad e \quad y_i = peso\ da\ pessoa_i$.

Dessa maneira, seguindo o que foi explicado anteriormente, tendo o conjunto de dados, Regressão Linear é utilizada para encontrar uma função h, dentro das funções lineares, que melhor consiga representar essa relação.

Generalizando essa ideia para um caso em que tenhamos d (dimensão) variáveis influenciando uma saída de maneira linear, sabemos que a função procurada será da forma:

$$h(\mathbf{x}_i) = w_0 + w_1 x_{i,1} + w_2 x_{i,2} + \ldots + w_d x_{i,d} = \mathbf{w}^T \tilde{\mathbf{x}}_i$$

Seguindo o exemplo 3.1, teríamos:

$$h(x_i) = w_0 + w_1 x_{i,1} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}^T \begin{bmatrix} 1 \\ x_{i,1} \end{bmatrix}$$

Então, o processo de aprendizado de nosso algoritmo consiste em encontrar o melhor vetor de pesos $\mathbf{w}$ que, dados os vetores de dados $\mathbf{x}_i$, produzirá o menor erro se comparado com as saídas $y_i,\ i = 1, \ldots, N$. Note que deixamos de tratar do exemplo 3.1 e estamos generalizamos o processo estudado.

Para mensurar esse erro, como comentado anteriormente, precisamos da função de perda. Desse modo, a definiremos da seguinte maneira:

$$\mathcal{L}(h) = \frac{1}{N}\sum_{i=1}^{N}(h(\tilde{\mathbf{x}}_i) - y_i)^2$$

Perceba que, como buscamos o vetor de pesos $\mathbf{w}$, podemos escrever

$$\mathcal{L}(h) = \mathcal{L}(\mathbf{w})$$

Essa expressão é o erro quadrático médio. Assim, para minimizar essa função há uma abordagem analítica e outra numérica.

3.1 1-Step Learning (Solução Analítica)

A função $\mathcal{L}(\mathbf{w})$ é convexa, isso quer dizer que, somente possui ponto de mínimo. Então, basta derivarmos a expressão e procurarmos por um ponto crítico. Faremos esse processo para um caso geral, com N pares de dados

$$(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$$

Sendo que, $\mathbf{x}_i\ i = 1, \ldots, N$ são vetores de dimensão d, representando os dados de entrada. Dessa maneira, como vamos lidar com muitos dados e de muitas dimensões, é interessante utilizar uma abordagem matricial para minimizar essa função:

$$\begin{bmatrix} h(\mathbf{x}_1) - y_1 \\ h(\mathbf{x}_2) - y_2 \\ \vdots \\ h(\mathbf{x}_N) - y_N \end{bmatrix} = \begin{bmatrix} w_0 + w_1 x_{11} + \ldots + w_d x_{1d} \\ w_0 + w_1 x_{21} + \ldots + w_d x_{2d} \\ \vdots \\ w_0 + w_1 x_{N1} + \ldots + w_d x_{Nd} \end{bmatrix} - \mathbf{y} = \begin{bmatrix} 1 & x_{11} & \ldots & x_{1d} \\ 1 & x_{21} & \ldots & x_{2d} \\ \vdots & & & \\ 1 & x_{N1} & \ldots & x_{Nd} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} - \mathbf{y} =$$

$$= \tilde{\mathbf{X}}\mathbf{w} - \mathbf{y}$$

Como desejamos a soma de todos os $(h(\mathbf{x}_i) - y_i)^2$, se utilizarmos a norma de matrizes, obtemos a raiz desse valor. Então, temos:

$$\begin{bmatrix} (h(\mathbf{x}_1) - y_1)^2 \\ (h(\mathbf{x}_2) - y_2)^2 \\ \vdots \\ (h(\mathbf{x}_N) - y_n)^2 \end{bmatrix} = \|\tilde{\mathbf{X}}\mathbf{w} - \mathbf{y}\|^2$$

Disso, podemos escrever a função de custo da seguinte forma:

$$\mathcal{L}(h) = \frac{1}{N}\|\tilde{\mathbf{X}}\mathbf{w} - \mathbf{y}\|^2$$

Então, para obter o vetor $\mathbf{w}$, basta minimizar a função, resolvendo

$$\nabla\mathcal{L}(h) = 0 \Rightarrow$$

$$\Rightarrow \nabla\mathcal{L}(h) = \frac{2}{N}\tilde{\mathbf{X}}^T(\tilde{\mathbf{X}}\mathbf{w} - \mathbf{y}) \Rightarrow$$

$$\Rightarrow \frac{2}{N}\tilde{\mathbf{X}}^T(\tilde{\mathbf{X}}\mathbf{w} - \mathbf{y}) = 0 \Rightarrow \tilde{\mathbf{X}}^T\tilde{\mathbf{X}}\mathbf{w} - \tilde{\mathbf{X}}^T\mathbf{y} = 0 \Rightarrow \tilde{\mathbf{X}}^T\tilde{\mathbf{X}}\mathbf{w} = \tilde{\mathbf{X}}^T\mathbf{y} \Rightarrow$$

$$\Rightarrow \mathbf{w} = (\tilde{\mathbf{X}}^T\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^T\mathbf{y}$$

Perceba que o cálculo do vetor $\mathbf{w}$ pode ser custoso. Primeiro efetuamos uma multiplicação de matrizes — $\tilde{\mathbf{X}}^T$ de tamanho $N \times (d+1)$ e $\tilde{\mathbf{X}}$ de tamanho $(d+1) \times N$ — depois invertemos uma matriz $(d+1) \times (d+1)$. Algoritmos para inverter matrizes de ordem n têm complexidade $O(n^3)$. Então, se N e d são muito grandes, por questões de eficiência computacional, não é interessante utilizar o método analítico de solução.

3.2 Solução Numérica

Como o método analítico mostrado pode apresentar problemas de eficiência, é válido aplicarmos métodos numéricos mais eficientes para otimizarmos a função de perda. Assim, vamos usar Gradiente Descendente. Como caminharemos para a direção oposta à do gradiente, lembre-se que devemos multiplicá-lo por -1.

3.2.1 Gradiente de $\mathcal{L}(\mathbf{w})$

Como estamos tratando do vetor de pesos $\mathbf{w}$, o gradiente de $\mathcal{L}(\mathbf{w})$ é dado por:

$$\nabla\mathcal{L}(\mathbf{w}) = \begin{bmatrix} \frac{\partial}{\partial w_0}\mathcal{L}(\mathbf{w}) & \frac{\partial}{\partial w_1}\mathcal{L}(\mathbf{w}) & \cdots & \frac{\partial}{\partial w_d}\mathcal{L}(\mathbf{w}) \end{bmatrix}$$

Tomando um elemento arbitrário $w_k,\ k = 0, \ldots, d$ de $\mathbf{w}$:

$$\frac{\partial}{\partial w_k}\mathcal{L}(\mathbf{w}) = \frac{\partial}{\partial w_k} - \frac{1}{N}\sum_{i=1}^{N}(h(\tilde{\mathbf{x}}_i) - y_i)^2 = \frac{\partial}{\partial w_k}\left(-\frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T\tilde{\mathbf{x}}_i - y_i)^2\right) \Rightarrow$$

$$\Rightarrow \frac{\partial}{\partial w_k}\mathcal{L}(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}\frac{\partial}{\partial w_k}(\mathbf{w}^T\tilde{\mathbf{x}}_i - y_i)^2 = -\frac{1}{N}\sum_{i=1}^{N}2(\mathbf{w}^T\tilde{\mathbf{x}}_i - y_i)\frac{\partial}{\partial w_k}(\mathbf{w}^T\tilde{\mathbf{x}}_i - y_i) \Rightarrow$$

$$\Rightarrow \frac{\partial}{\partial w_k}\mathcal{L}(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}2(\mathbf{w}^T\tilde{\mathbf{x}}_i - y_i)\left(\frac{\partial}{\partial w_k}\mathbf{w}^T\tilde{\mathbf{x}}_i - \frac{\partial}{\partial w_k}y_i\right)$$

Perceba que a expressão $\dfrac{\partial}{\partial w_k}\mathbf{w}^T\tilde{\mathbf{x}}_i = x_{ik}$:

$$\frac{\partial}{\partial w_k}\mathbf{w}^T\tilde{\mathbf{x}}_i = \frac{\partial}{\partial w_k}\left(w_0 + w_1x_{i1} + \ldots w_kx_{ik} + \ldots + w_dx_{id}\right) \Rightarrow$$

$$\Rightarrow \frac{\partial}{\partial w_k}\mathbf{w}^T\tilde{\mathbf{x}}_i = \frac{\partial}{\partial w_k}w_0 + \frac{\partial}{\partial w_k}w_1x_{i1} + \ldots + \frac{\partial}{\partial w_k}w_kx_{ik} + \ldots + \frac{\partial}{\partial w_k}w_dx_{id} \Rightarrow$$

$$\Rightarrow \frac{\partial}{\partial w_k}\mathbf{w}^T\tilde{\mathbf{x}}_i = 0 + 0 + \ldots + \frac{\partial}{\partial w_k}w_kx_{ik} + \ldots + 0 = x_{ik}$$

Voltando a $\dfrac{\partial}{\partial w_k}\mathcal{L}(\mathbf{w})$:

$$\frac{\partial}{\partial w_k}\mathcal{L}(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}2(\mathbf{w}^T\tilde{\mathbf{x}}_i - y_i)x_{ik} = -\frac{2}{N}\sum_{i=1}^{N}(\mathbf{w}^T\tilde{\mathbf{x}}_i - y_i)x_{ik} \Rightarrow$$

$$\frac{\partial}{\partial w_k}\mathcal{L}(\mathbf{w}) = \frac{2}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\tilde{\mathbf{x}}_i)x_{ik} = \frac{2}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)x_{ik}$$

Generalizando a expressão encontrada para os demais valores que k assume, encontramos o valor de $\nabla\mathcal{L}(\mathbf{w})$.

3.2.2 Gradiente Descendente

Desse modo, como obtemos o valor do gradiente da função de perda, temos que o passo de atualização de cada elemento do vetor de pesos na técnica do gradiente descendente segue o seguinte padrão:

$$w_k(t+1) \leftarrow w_k(t) - \eta\left(\frac{2}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)x_{ik}\right)$$

3.3 Implementação

No sentido de testar o algoritmo de Regressão Linear, usaremos um dataset simples, composto de cerca de 50 registros, compilando informações sobre residências do mercado imobiliário de Oregon. Há informações sobre o preço, quantidade de quartos e área.

Esse trabalho de implementação foi realizado utilizando o software Google Collab, e IPython Notebooks. Pode-se acessar tal trabalho pelo link: NOTEBOOK REGRESSÃO LINEAR

4 Regressão Logística

A Regressão Logística atua de forma semelhante à Regressão Linear, que foi discutida anteriormente. A diferença entre essas duas técnicas é que enquanto, com Regressão Linear, queremos prever valores contínuos na reta real — isto é, valores que estejam contidos no intervalo $[-\infty, +\infty]$ — com Regressão Logística, desejamos prever valores que estejam contidos no intervalo $[0, 1]$, que (como mostraremos adiante) podem ser interpretados como probabilidades.

Como queremos prever probabilidades utilizando o modelo linear, é importante perceber que estamos assumindo que as probabilidades dos eventos, tem uma relação linear com os dados de entrada.

Exemplo 4.1 *Suponha que queiramos prever a probabilidade de um senhor de 80 anos ter um infarto. Nesse sentido, assumimos que a probabilidade de uma pessoa sofrer um infarto cresce de maneira que se assemelha à algo linear, a qual aplicaremos o modelo linear no futuro para modelar.*

Perceba que o modelo linear desenvolvido até agora consiste na obtenção de uma expressão linear que relacione adequadamente valores de entrada com suas respectivas saídas. Dessa forma, nos lembremos que as expressões produzidas são funções da seguinte forma: $f : \mathbb{R}^d \to \mathbb{R}$; em que temos uma entrada de dados de dimensão d levando à um valor escalar y.

Nesse sentido, fica evidente que esse modelo linear não pode ser aplicado diretamente em uma situação em que tentamos descrever algo que se assemelhe à uma função $f : \mathbb{R}^d \to [0, 1]$. Por isso, devemos fazer mudanças no domínio de nossa função para que possamos utilizar coerentemente esse modelo linear para abordar o problema.

4.1 Odds

Como queremos prever probabilidades e a natureza dessa classe de valores (se tratar de um número entre 0 e 1) é o que está atrapalhando a utilização do modelo linear, o caminho natural para contornar esse problema é tentar representar probabilidades de outras maneiras. Ou seja, mapear os valores de probabilidades para valores que possam assumir qualquer valor na reta real.

Desse modo, o conceito de Odds "chance" é útil. Seja p a probabilidade de um evento ocorrer. Então, por definição, a chance desse evento ocorrer é dada por:

$$\text{Odds}(p) = \frac{p}{1-p}$$

Perceba que, se $p = 0$, então $\text{Odds}(p) = 0$. Agora, se $p = 1$, $\text{Odds}(p)$ tende a $+\infty$ disso, $\text{Odds}(p) \in [0, +\infty]$. Nesse momento, apesar de termos feito progresso na tarefa de representar as probabilidades no intervalo $[-\infty, +\infty]$, temos alguns problemas.

Observe o seguinte ponto:

- **A medida de Odds apresenta um comportamento indesejado quanto a relação linear dos dados de entrada com a saída.**

Aqui está um problema definitivo na ideia de tentar utilizar Odds para prever probabilidades de maneira linear. Isso por que, assumindo que as probabilidades tenham o comportamento linear que foi comentado, é importante que a função que usaremos para tentar modelar as probabilidades tenha algum tipo de simetria. Perceba que, enquanto $\text{Odds}(p)$ tal que $p < 0.5$ fica restrito ao pequeno intervalo de $]0, 1[$, $\text{Odds}(p)$ com $p > 0.5$ pode ocupar valores em $]1, +\infty[$. Ainda, esse problema está relacionado, também, ao fato de que desejamos que o valor previsto possa assumir qualquer valor na reta real. Perceba também, que, se houvesse simetria, esse problema da falta de valores negativos que vão até $-\infty$ seria resolvido.

Então, esse comportamento nada linear estraga as pretensões de usarmos $\text{Odds}(p)$ para tentar prever p.

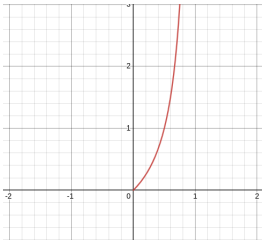

Figura 1: Gráfico de Odds(x)

4.2 logOdds

Para resolver esse empecilho, usamos:

$$\log(\text{Odds}(p)) = \log\left(\frac{p}{1-p}\right) = \log(p) - \log(1-p)$$

Aplicando essa transformação, temos um ponto a partir do qual a função apresenta a simetria requisitada e a imagem da função se torna $[-\infty, +\infty]$. Note que, se a função é simétrica a partir de algum ponto, temos:

$$\begin{aligned}
|f(x_0) - f(x_0 + h)| &= |f(x_0) - f(x_0 - h)| \Rightarrow \\
\Rightarrow |\log(x_0) - \log(1 - x_0) - (\log(x_0 + h) - \log(1 - x_0 - h))| &= \\
= |\log(x_0) - \log(1 - x_0) - (\log(x_0 - h) - \log(1 - x_0 + h))| &\Rightarrow \\
\Rightarrow \log(x_0 + h) + \log(1 - x_0 - h) &= \log(x_0 - h) + \log(1 - x_0 + h) \Rightarrow \\
\Rightarrow \log((x_0 + h)(1 - x_0 - h)) &= \log((x_0 - h)(1 - x_0 + h)) \Rightarrow \\
\Rightarrow (x_0 + h)(1 - x_0 - h) &= (x_0 - h)(1 - x_0 + h) \Rightarrow \\
\Rightarrow h - 2x_0h = -h + 2x_0h \Rightarrow 1 - 2x_0 = -1 + 2x_0 &\Rightarrow x_0 = 0.5
\end{aligned}$$

Observe o gráfico na figura 2 como o ponto de simetria do gráfico é, justamente, o ponto $x_0 = 0.5$ encontrado.

Perceba agora que, quando usamos $\log(\text{Odds}(p))$ ocorre também a contemplação dos valores negativos e isso é uma consequência da simetria que recuperamos! Nesse sentido, se queremos prever probabilidades utilizando o modelo linear, a ideia é prever $\log(\text{Odds}(p))$ e, então, converter o valor previsto para o seu respectivo valor de probabilidade. Problema resolvido!

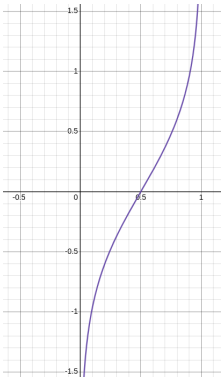

Figura 2: Gráfico de $\log(\text{Odds}(x))$

Agora, como obter p de $\log(\text{Odds}(p))$? Para isso, temos o que se segue:

$$\log(\text{Odds}(p)) = \ln\left(\frac{p}{1-p}\right) \approx \mathbf{w}^T\tilde{\mathbf{x}} \Rightarrow$$

$$\Rightarrow e^{\mathbf{w}^T\tilde{\mathbf{x}}} = \frac{p}{1-p} \Rightarrow p = (1-p)e^{\mathbf{w}^T\tilde{\mathbf{x}}} \Rightarrow p = e^{\mathbf{w}^T\tilde{\mathbf{x}}} - e^{\mathbf{w}^T\tilde{\mathbf{x}}}p \Rightarrow$$

$$\Rightarrow p + e^{\mathbf{w}^T\tilde{\mathbf{x}}}p = e^{\mathbf{w}^T\tilde{\mathbf{x}}} \Rightarrow p(1 + e^{\mathbf{w}^T\tilde{\mathbf{x}}}) = e^{\mathbf{w}^T\tilde{\mathbf{x}}} \Rightarrow p = \frac{e^{\mathbf{w}^T\tilde{\mathbf{x}}}}{1 + e^{\mathbf{w}^T\tilde{\mathbf{x}}}} \Rightarrow$$

$$p = \frac{1}{\frac{1}{e^{\mathbf{w}^T\tilde{\mathbf{x}}}} + 1} \Rightarrow p = \frac{1}{1 + e^{-(\mathbf{w}^T\tilde{\mathbf{x}})}} = \hat{y}$$

A expressão de $\hat{y}$ é definida como "Curva Logística".

4.2.1 Função Sigmoide

A função sigmoide é uma função matemática cujo gráfico se assemelha ao formato da letra "s".

Essa função pode ser definida como:

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

Dessa forma, a expressão para prever probabilidades (que obtivemos para a Regressão Logística) é tal que $\sigma\left(\mathbf{w}^T\tilde{\mathbf{x}}\right)$.

4.3 Função de Perda ou Custo

Pensando em Regressão Linear, poderíamos pensar em utilizar a mesma estratégia (erro quadrático médio) como função de perda, afinal, as ideias

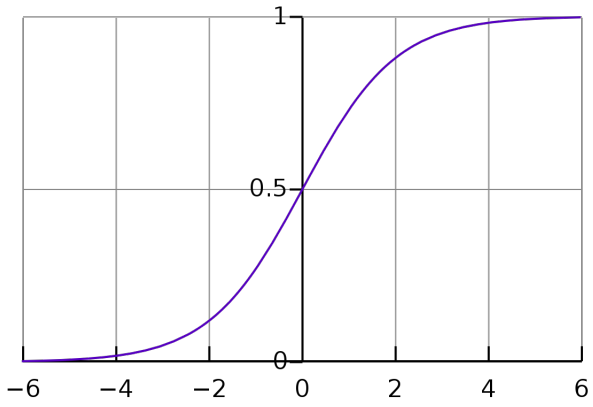

Figura 3: Função Sigmoide

envolvidas fazem muito sentido. Contudo, temos um problema que impede esse uso. Em Regressão Logística, a função:

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T \tilde{\mathbf{x}})}}$$

é não linear. Nesse sentido, se tentarmos utilizar o método dos quadrados mínimos, produziremos uma função não convexa. Então, a procura de um ponto mínimo se torna uma tarefa bastante capciosa. Além disso, como os valores que y assumem são somente 1 e 0, os valores de erro do modelo estarão sempre no intervalo $[0, 1]$, então necessitaríamos de muita precisão de ponto flutuante ao longo das iterações.

Assim, a alternativa que iremos utilizar (existem outras) é a da Máxima Verossimilhança.

4.3.1 Verossimilhança (Likelihood)

Primeiramente, a função de verossimilhança é corresponde à distribuição de probabilidade conjunta — isto é, o comportamento simultâneo de probabilidades — de um conjunto de dados em função dos parâmetros do modelo estatístico em questão.

$$P(\mathbf{X}|\theta)$$

Pensando no caso de Regressão Logística, em que interpretamos a saída como probabilidades (o que se generaliza para problemas de classificação em geral), podemos pensar na função de Verossimilhança como:

$$P(\mathbf{X}|\mathbf{w})$$

Onde o conjunto de dados $\mathbf{X}$ se trata do conjunto dos pares de dados $(\mathbf{x}_i, y_i),\ i = 1, \ldots, N$. Nesse sentido, podemos representar tais pares da seguinte maneira:

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N) = X_1, X_2, \ldots, X_N$$

Se tratando de probabilidade conjunta, assumimos que os pares de dados (que compõe uma observação cada) são independentes entre si. Por isso:

$$P(\mathbf{X}|\mathbf{w}) = P(X_1, \ldots, X_N|\mathbf{w}) = P(X_1|\mathbf{w}) \cdots P(X_N|\mathbf{w}) = \prod_{i=1}^{N} P(X_i|\mathbf{w})$$

Note que $P(X_i|\mathbf{w})$ significa a probabilidade de observar o par de dados $(\mathbf{x}_i, y_i)$ dado o vetor de pesos $\mathbf{w}$.

Agora, perceba que $\sigma(\mathbf{w}^T\tilde{\mathbf{x}}_i)$ é a probabilidade de observarmos $y_i = 1$ dado o vetor de entrada $\mathbf{x}_i$ e o vetor de pesos $\mathbf{w}$. De maneira análoga, $1 - \sigma(\mathbf{w}^T\tilde{\mathbf{x}}_i)$ é a probabilidade de observarmos $y_i = 0$.

Nesse sentido, se $P(X_i|\mathbf{w})$ é tal que $y_i = 1$, então $P(X_i|\mathbf{w}) = \sigma(\mathbf{w}^T\tilde{\mathbf{x}}_i) = \hat{y}_i$ e se $P(X_i|\mathbf{w})$ é tal que $y_i = 0$, então $P(X_i|\mathbf{w}) = 1 - \sigma(\mathbf{w}^T\tilde{\mathbf{x}}_i) = 1 - \hat{y}_i$.

Disso, podemos separar o produtório $\prod_{i=1}^{N} P(X_i|\mathbf{w})$ em dois casos:

$$\prod_{i=1}^{N} P(X_i|\mathbf{w}) = \prod_{\substack{i=1 \\ \text{se } y_i=1}}^{N} \hat{y}_i \cdot \prod_{\substack{i=1 \\ \text{se } y_i=0}}^{N} 1 - \hat{y}_i$$

Repare que, devido às propriedades de exponenciação, temos:

$$\prod_{i=1}^{N} P(X_i|\mathbf{w}) = \prod_{\substack{i=1 \\ \text{se } y_i=1}}^{N} \hat{y}_i \cdot \prod_{\substack{i=1 \\ \text{se } y_i=0}}^{N} 1 - \hat{y}_i = \prod_{i=1}^{N} \hat{y}_i^{y_i} \cdot \prod_{i=1}^{N} (1 - \hat{y}_i)^{1-y_i}$$

Então, podemos unir os dois produtórios:

$$P(\mathbf{X}|\mathbf{w}) = \prod_{i=1}^{N} P(X_i|\mathbf{w}) = \prod_{i=1}^{N} \hat{y}_i^{y_i}(1 - \hat{y}_i)^{1-y_i}$$

E essa é a expressão obtida para a função de verossimilhança em Regressão Logística.

4.3.2 Máxima Verossimilhança

Podemos reescrever a fórmula obtida para a verossimilhança de modo que ela nos sirva melhor:

$$P(\mathbf{X}|\mathbf{w}) = \prod_{i=1}^{N} \hat{y}_i^{y_i}(1-\hat{y}_i)^{1-y_i} =$$

$$= \prod_{i=1}^{N} \left(\frac{1}{1+e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}\right)^{y_i} \left(1-\frac{1}{1+e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}\right)^{1-y_i} =$$

$$= \prod_{i=1}^{N} \left(\frac{1}{1+e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}\right)^{y_i} \left(\frac{e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}{1+e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}\right)^{1-y_i} = l(\mathbf{w})$$

Note que, todo o intuito dessa seção é prever/estimar probabilidades. Nesse sentido, é evidente que a probabilidade presente na expressão ($\hat{y}_i$) é a que estamos prevendo. E, assim, está em função dos dados de entrada. Além disso, a expressão que encontramos para $\hat{y}_i$ está em termos do nosso vetor de pesos $\mathbf{w}$ — tudo volta ao modelo linear!

Agora, queremos encontrar os valores de $\mathbf{w}$ que maximizam essa função. Mas, perceba que maximizar $l(\mathbf{w})$ é equivalente a maximizar $\log(l(\mathbf{w}))$. Assim, temos:

4.3.3 Log-Likelihood

$$\mathcal{L}(\mathbf{w}) = \ln\left(\prod_{i=1}^{N}\left(\frac{1}{1+e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}\right)^{y_i}\left(\frac{e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}{1+e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}\right)^{1-y_i}\right) \Rightarrow$$

$$\Rightarrow \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} y_i \ln\left(\frac{1}{1+e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}\right) + (1-y_i)\ln\left(\frac{e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}{1+e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}\right) \Rightarrow$$

$$\Rightarrow \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} y_i\left(\ln\left(\frac{1}{1+e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}\right) - \ln\left(\frac{e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}{1+e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}\right)\right) + \ln\left(\frac{e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}{1+e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}}\right) \Rightarrow$$

$$\Rightarrow \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} y_i \ln\left(e^{\mathbf{w}^T\tilde{\mathbf{x}}_i}\right) + \ln\left(\frac{1}{1+e^{\mathbf{w}^T\tilde{\mathbf{x}}_i}}\right) \Rightarrow$$

$$\Rightarrow \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} y_i\mathbf{w}^T\tilde{\mathbf{x}}_i - \ln\left(1+e^{\mathbf{w}^T\tilde{\mathbf{x}}_i}\right)$$

Além disso, uma prática útil é transformar essa medida de erro relativa à todos os exemplos em um erro médio:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} y_i\mathbf{w}^T\tilde{\mathbf{x}}_i - \ln\left(1+e^{\mathbf{w}^T\tilde{\mathbf{x}}_i}\right)$$

Desse modo, para maximizar $\mathcal{L}(\mathbf{w})$ usaremos o método de otimização Gradiente Descendente (há diversos outros).

4.4 Otimização

4.4.1 Gradiente de $\mathcal{L}(\mathbf{w})$

$$\nabla\mathcal{L}(\mathbf{w}) = \left[\frac{\partial}{\partial w_0}\mathcal{L}(\mathbf{w}) \quad \frac{\partial}{\partial w_1}\mathcal{L}(\mathbf{w}) \quad \cdots \quad \frac{\partial}{\partial w_d}\mathcal{L}(\mathbf{w})\right]$$

Vamos observar então, para um k arbitrário, $\frac{\partial}{\partial w_k}\mathcal{L}(\mathbf{w})$:

$$\frac{\partial}{\partial w_k}\mathcal{L}(\mathbf{w}) = \frac{\partial}{\partial w_k}\frac{1}{N}\sum_{i=1}^{N} y_i \mathbf{w}^T\tilde{\mathbf{x}}_i - \ln\left(1+e^{\mathbf{w}^T\tilde{\mathbf{x}}_i}\right) \Rightarrow$$

$$\Rightarrow \frac{\partial}{\partial w_k}\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial}{\partial w_k}\left(y_i\mathbf{w}^T\tilde{\mathbf{x}}_i\right) - \frac{\partial}{\partial w_k}\left(\ln\left(1+e^{\mathbf{w}^T\tilde{\mathbf{x}}_i}\right)\right)$$

$$\frac{\partial}{\partial w_k}\left(\ln\left(1+e^{\mathbf{w}^T\tilde{\mathbf{x}}_i}\right)\right) = \frac{1}{1+e^{\mathbf{w}^T\tilde{\mathbf{x}}_i}}\frac{\partial}{\partial w_k}(1+e^{\mathbf{w}^T\tilde{\mathbf{x}}_i}) = \frac{e^{\mathbf{w}^T\tilde{\mathbf{x}}_i}}{1+e^{\mathbf{w}^T\tilde{\mathbf{x}}_i}}\frac{\partial}{\partial w_k}\mathbf{w}^T\tilde{\mathbf{x}}_i$$

$$\frac{\partial}{\partial w_k}\left(y_i\mathbf{w}^T\tilde{\mathbf{x}}_i\right) = y_i\frac{\partial}{\partial w_k}\mathbf{w}^T\tilde{\mathbf{x}}_i$$

Relembrando que $\frac{\partial}{\partial w_k}\mathbf{w}^T\tilde{\mathbf{x}}_i = x_{ik}$ Então,

$$\frac{\partial}{\partial w_k}\left(\ln\left(1+e^{\mathbf{w}^T\tilde{\mathbf{x}}_i}\right)\right) = \frac{x_k e^{\mathbf{w}^T\tilde{\mathbf{x}}_i}}{1+e^{\mathbf{w}^T\tilde{\mathbf{x}}_i}} = x_{ik}\frac{1}{1+e^{-(\mathbf{w}^T\tilde{\mathbf{x}}_i)}} = x_{ik}\hat{y}_i$$

$$\frac{\partial}{\partial w_k}\left(y_i\mathbf{w}^T\tilde{\mathbf{x}}_i\right) = y_i x_{ik}$$

$$\frac{\partial}{\partial w_k}\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} y_i x_{ik} - x_{ik}\hat{y}_i = \frac{1}{N}\sum_{i=1}^{N} x_{ik}\left(y_i - \hat{y}_i\right)$$

Fazendo isso para todo $k = 0, \ldots, d$, obtemos $\nabla\mathcal{L}(\mathbf{w})$.

4.4.2 Gradiente Descendente

Como já obtemos o valor de $\nabla\mathcal{L}(\mathbf{w})$, utilizando a técnica do Gradiente Descendente para otimizar a função de perda, chegamos ao passo de atualização do vetor de pesos $\mathbf{w}$ na Regressão Logística:

$$w_k(t+1) \leftarrow w_k(t) - \eta \left(\frac{1}{N} \sum_{i=1}^{N} x_{ik} \left(y_i - \frac{1}{1 + e^{-(\mathbf{w}^T \tilde{\mathbf{x}}_i)}} \right) \right)$$

4.5 Implementação

Agora, como já construímos base teórica suficiente sobre a Regressão Logística, partimos para a implementação desse modelo.

Para testar o modelo, usamos um dataset genérico encontrado na internet que possui dados de usuários de redes sociais. Esse dataset relaciona tais dados com o fato dos usuários terem efetuado ou não compras através de uma empresa de anúncios da plataforma.

Seguindo a linha da implementação de Regressão Linear, é possível conferir o trabalho feito em um Notebook Python por meio do seguinte link: NOTEBOOK REGRESSÃO LOGÍSTICA

5 Regressão Logística Multinomial

Até agora, em relação à Regressão Logística, estudamos cenários em que a variável dependente (y) assumiu valores binários. Isso porque, tratamos esse tipo de regressão como um algoritmo de classificação com 2 classes. Quando há mais de duas saídas possíveis, caímos no problema da Regressão Logística Multinomial — que também recebe o nome de Regressão Softmax.

Podemos elencar diversos exemplos que recaem na situação descrita:

1. Dados hábitos e costumes, em qual candidato uma pessoa irá votar?
2. Qual cidade brasileira seria o destino de uma viagem, dado preferências pessoais e orçamento?

Perceba que ainda temos dados de entrada $\mathbf{x}$ (variáveis independentes) e queremos obter um resultado dentre uma lista de opções. Nesse sentido, é

importante ressaltar que os resultados da regressão não podem ser ordenados. Isto é, ao se tratar do segundo (2) item, não faria sentido ordenarmos as cidades (possíveis resultados) de acordo com alguma ordem de importância, por exemplo.

Assim, para tratar de questões dessa natureza, que surgem ao considerarmos problemas de classificação multiclasses, é conveniente o uso de outra forma de representação da saída.

5.1 One-hot Encoding

Até agora, por padrão, estávamos utilizando o chamado "Label encoding". Nesse caso, utilizamos números para representar as classes em questão. Como feito no seguinte exemplo:

Exemplo 5.1 *Considere que desejamos construir um modelo de Machine Learning para dizer se uma imagem de um certo animal é um cachorro, um gato ou uma ave. Dessa forma, poderíamos representar tais dados (cada par, de entrada e rótulo) em um conjunto de treinamento da seguinte maneira:*

$$(\mathbf{x}, y)$$

Onde $\mathbf{x}$ *é uma imagem e* y *é um valor que representa a classe a qual essa imagem pertence, como na seguinte tabela:*

Animal	Rótulo de classe
Gato	1
Cachorro	2
Ave	3

Contudo, como comentado, surge a necessidade de outra forma de representação de dados. O chamado One-hot encoding é uma maneira mais binarizada de representação de múltiplas classes.

Nessa abordagem, separamos os possíveis valores dos resultados em colunas binárias que indicam a correspondência de uma entrada à uma das possíveis classes. Observe em um exemplo:

Exemplo 5.2 *Suponha que, dado a velocidade em que um certo animal se locomove, queiramos decidir a qual espécie ele pertence. Desse modo, poderíamos receber uma tabela que relacionasse velocidades de locomoção à*

Animal	Velocidade (km/h)
Guepardo	105
Elefante	40
Pastor-alemão	48
Falcão	390

espécies (seguindo o padrão label encoding):

Esses dados poderiam, então ser representados como pares de dados da seguinte maneira:

$$(x_i, y_i)\ i = 1, 2, 3, 4$$

$$(105, \textit{Guepardo}), (40, \textit{Elefante}), (48, \textit{Pastor-alemão}), (390, \textit{Falcão})$$

Por outro lado, a representação desses dados com One-hot encoding seria:

Velocidade (km/h)	Guepardo	Elefante	Pastor-alemão	Falcão
105	1	0	0	0
40	0	1	0	0
48	0	0	1	0
390	0	0	0	1

A qual pode ser representada, pelos pares de dados:

$$(x_i, \boldsymbol{y}_i)\ i = 1, 2, 3, 4$$

$$(105, \text{ [1 0 0 0] }), (40, \text{ [0 1 0 0] }), (48, \text{ [0 0 1 0] }), (390, \text{ [0 0 0 1] })$$

5.2 Classificadores Essencialmente Multiclasses

Para resolver o problema de classificação com mais de duas classes, existem abordagens que combinam classificadores binários para atingir o resultado esperado. Contudo, nesse texto, queremos, efetivamente, calcular:

$$P(y = j|\mathbf{x}),\ \ j = 1, 2, \ldots, K$$

Para fazer isso, utilizaremos a função Softmax:

$$\hat{p}_j = P(y = j|\mathbf{x}) = \frac{e^{\mathbf{w}_j^T \tilde{\mathbf{x}}}}{\sum_{q=1}^{K} e^{\mathbf{w}_q^T \tilde{\mathbf{x}}}}, \quad j = 1, 2, \ldots, K$$

5.3 Função Softmax

A função Softmax recebe um vetor de números reais e retorna um vetor de números reais no intervalo $[0, 1]$.

Observe que o vetor $\mathbf{x}$ é o vetor de entrada que está sendo analisado e sobre o qual queremos prever o resultado ($\hat{y}$). Tratando agora sobre o $\tilde{\mathbf{x}}$, esse vetor é, tão somente, o vetor $\mathbf{x}$ adicionado do valor 1 em sua primeira entrada. Então,

$$\tilde{\mathbf{x}} = (1, x_1, x_2, \cdots, x_d)$$

Além disso, teremos K vetores de pesos $\mathbf{w}$. Cada um deles com d componentes (mais o threshold w_0, totalizando $d + 1$ elementos). Que possibilitam a multiplicação $\mathbf{w}_j^T \tilde{\mathbf{x}}$:

$$\mathbf{w}_j^T \tilde{\mathbf{x}} = w_{j,0} + w_{j,1} x_1 + \ldots + w_{j,d} x_d$$

Dessas considerações, podemos perceber alguns detalhes importantes.

1. $0 \leq \hat{p}_j \leq 1$
2. $\sum_{j=1}^{K} \hat{p}_j = 1$

$$\sum_{j=1}^{K} \hat{p}_j = \frac{e^{\mathbf{w}_1^T \tilde{\mathbf{x}}}}{\sum_{q=1}^{K} e^{\mathbf{w}_q^T \tilde{\mathbf{x}}}} + \frac{e^{\mathbf{w}_2^T \tilde{\mathbf{x}}}}{\sum_{q=1}^{K} e^{\mathbf{w}_q^T \tilde{\mathbf{x}}}} + \cdots + \frac{e^{\mathbf{w}_K^T \tilde{\mathbf{x}}}}{\sum_{q=1}^{K} e^{\mathbf{w}_q^T \tilde{\mathbf{x}}}} = \frac{\sum_{j=1}^{K} e^{\mathbf{w}_j^T \tilde{\mathbf{x}}}}{\sum_{q=1}^{K} e^{\mathbf{w}_q^T \tilde{\mathbf{x}}}} = 1$$

3. A classe escolhida será $\arg\max \{\mathbf{w}_j^T \tilde{\mathbf{x}}\} = \arg\max \{\hat{p}_j\}$. Ou seja, o vetor de pesos que produzir maior valor a partir da entrada estudada ($\tilde{\mathbf{x}}$) será o escolhido e seu índice j representa a classe escolhida.

5.3.1 Softmax e Sigmoide

A semelhança entre a expressão da função Softmax com a Sigmoide, definida na seção de Regressão Logística, é evidente. Isso ocorre pois a função Softmax corresponde à uma generalização da função Sigmoide para K dimensões. Isso se torna claro na medida em que podemos obter a expressão da função Softmax a partir da Curva Logística e vice versa. Em suma, a função Logística pode ser interpretada como a função Softmax com $K = 2$.

Para enxergar isso melhor, podemos usar o mesmo raciocínio desenvolvido ao longo da seção destinada à Regressão Logística, onde assumimos que conseguimos relacionar $\log(Odds(p))$ utilizando o modelo linear $\mathbf{w}^T\mathbf{x}$ para obter $\hat{y} = p$ usando as variáveis de entrada $\mathbf{x}$.

Dessa forma, consideramos que, no caso multinomial, o conceito de *Odd* será usar a razão de $P(y = j|\mathbf{x})$ com $P(y = K|\mathbf{x})$ da seguinte maneira:

$$\log\left(\frac{P(y=j|\mathbf{x})}{P(y=K|\mathbf{x})}\right) = \mathbf{w}_j^T\tilde{\mathbf{x}},\ j = 1, \ldots, K$$

Disso, temos:

$$\frac{P(y=j|\mathbf{x})}{P(y=K|\mathbf{x})} = e^{\mathbf{w}_j^T\tilde{\mathbf{x}}} \Rightarrow P(y=j|\mathbf{x}) = e^{\mathbf{w}_j^T\tilde{\mathbf{x}}}P(y=K|\mathbf{x})$$

Agora, usamos o fato esses valores de probabilidade devem somar 1:

$$\sum_{q=1}^{K} P(y=q|\mathbf{x}) = 1 \Rightarrow \sum_{q=1}^{K} e^{\mathbf{w}_q^T\tilde{\mathbf{x}}}P(y=K|\mathbf{x}) = 1 \Rightarrow$$

$$\Rightarrow P(y=K|\mathbf{x})\sum_{q=1}^{K} e^{\mathbf{w}_q^T\tilde{\mathbf{x}}} = 1 \Rightarrow P(y=K|\mathbf{x}) = \frac{1}{\sum_{q=1}^{K} e^{\mathbf{w}_q^T\tilde{\mathbf{x}}}}$$

Agora, multiplicamos ambos os lados da equação por $e^{\mathbf{w}_j^T\tilde{\mathbf{x}}}$:

$$e^{\mathbf{w}_j^T\tilde{\mathbf{x}}}P(y=K|\mathbf{x}) = \frac{e^{\mathbf{w}_j^T\tilde{\mathbf{x}}}}{\sum_{q=1}^{K} e^{\mathbf{w}_q^T\tilde{\mathbf{x}}}} \Rightarrow P(y=j|\mathbf{x}) = \frac{e^{\mathbf{w}_j^T\tilde{\mathbf{x}}}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^T\tilde{\mathbf{x}}}}$$

5.3.2 Função de Perda

Como a função Softmax se assemelha à função Sigmoide, a função de perda utilizada em Regressão Softmax também não será tão diferente da utilizada em Regressão Logística. Nesse sentido, usaremos os conceitos de Verossimilhança. Temos que a função de Verossimilhança no caso multiclasses é

$$P(\mathbf{X}|\mathbf{W})$$

Como a função de Verossimilhança é a distribuição conjunta dos dados observados, temos:

$$P(X^{(1)}, X^{(2)}, \ldots, X^{(N)}|\mathbf{W})$$

Onde N é o número de dados no conjunto de treinamento, $\mathbf{W}$ é o conjunto dos vetores de pesos $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K\}$ e $X^{(i)}$ corresponde ao par de dados $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$. Perceba, também, a mudança de notação. Agora, estamos usando índices sobrescritos. De modo que, $\mathbf{x}^{(i)}$ é o exemplo de treinamento i e seu rótulo é $y^{(i)}$.

Partindo do pressuposto de que os dados são independentes:

$$P(\mathbf{X}|\mathbf{W}) = P(X^{(1)}|\mathbf{W}) \cdot \ldots \cdot P(X^{(N)}|\mathbf{W}) =$$

$$= \prod_{i=1}^{N} P(X^{(i)}|\mathbf{W})$$

Mas agora, qual é o valor de $P(X^{(i)}|\mathbf{W})$?

Como, se tratando de Softmax, utilizando o exemplo de treinamento i, a saída da função é o vetor

$$\hat{y}^{(i)} = \begin{bmatrix} \hat{y}_1^{(i)} \\ \hat{y}_2^{(i)} \\ \vdots \\ \hat{y}_N^{(i)} \end{bmatrix}$$

Em que $\hat{y}_j^{(i)}$ é a probabilidade (predita pelo modelo) de que $\mathbf{x}^{(i)}$ pertença à classe j e lembrando que o vetor $y^{(i)}$ é One-Hot Encoded:

$$y^{(i)} = \begin{bmatrix} y_1^{(i)} \\ y_2^{(i)} \\ \vdots \\ y_N^{(i)} \end{bmatrix}$$

Onde, $y_j^{(i)} = 1$ se $\mathbf{x}^{(i)}$ é da classe j e $y_j^{(i)} = 0$ caso contrário, podemos utilizar tais valores para construir a seguinte função — que avaliará a saída relativa ao rótulo verdadeiro do exemplo i:

$$P(X^{(i)}|\mathbf{W}) = \prod_{q=1}^{K} {\hat{y}_q^{(i)}}^{y_q^{(i)}}$$

Considere que $\mathbf{x}^{(i)}$ seja da classe j. Dessa forma, teremos:

$$P(X^{(i)}|\mathbf{W}) = \prod_{q=1}^{K} {\hat{y}_q^{(i)}}^{y_q^{(i)}} = \hat{y}_j^{(i)}$$

Portanto, se o modelo obtiver um valor alto para $\hat{y}_j^{(i)}$, deveríamos obter um valor alto de verossimilhança e isso é justamente o que ocorre, pois teremos teremos um valor alto de $P(X^{(i)}|\mathbf{W})$.

Por outro lado, se o modelo nos entregar um valor baixo para $\hat{y}_j^{(i)}$, saberemos que essa foi uma má previsão (baixa verossimilhança), pois o valor de $P(X^{(i)}|\mathbf{W})$ será baixo.

Assim, passeando por todos os dados de exemplos no conjunto de treinamento, se os valores previstos $\hat{y}_j^{(i)},\ i = 1, \ldots, N\ \ j = 1, \ldots, K$ se assemelharem aos rótulos $y_j^{(i)}$, então o produtório:

$$P(\mathbf{X}|\mathbf{W}) = \prod_{i=1}^{N} \prod_{q=1}^{K} \hat{y}_q^{(i)^{y_q^{(i)}}}$$

terá valor alto. Ou seja, alta Verossimilhança. E o oposto também vale. Então, temos definida a nossa função de custo: $l(\mathbf{W}) = \prod_{i=1}^{N} \prod_{q=1}^{K} \hat{y}_q^{(i)^{y_q^{(i)}}}$

Sabemos que maximizar $l(\mathbf{W})$ equivale à maximizar $\log(l(\mathbf{W}))$ e, também à minimizar $\mathcal{L}(\mathbf{W}) = -\log(l(\mathbf{W}))$, também continuaremos com a convenção de utilizar o erro médio. Disso, temos:

$$\mathcal{L}(\mathbf{W}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{q=1}^{K} y_q^{(i)} \ln(\hat{y}_q^{(i)})$$

5.3.3 Otimização

Como iremos otimizar a função de perda utilizando o método do gradiente descendente, devemos calcular $\nabla\mathcal{L}(\mathbf{W}) = \left[\frac{\partial}{\partial \mathbf{w}_1}\mathcal{L}(\mathbf{W}) \quad \frac{\partial}{\partial \mathbf{w}_2}\mathcal{L}(\mathbf{W}) \quad \cdots \quad \frac{\partial}{\partial \mathbf{w}_K}\mathcal{L}(\mathbf{W})\right]$ Onde $\boldsymbol{w_g},\ g = 1, \ldots, K$ é um dos K vetores de pesos com $d + 1$ elementos cada.

5.3.3.1 Gradiente de $\mathcal{L}(\mathbf{W})$

Queremos diferenciar $\mathcal{L}(\mathbf{W})$ em relação ao vetor $\mathbf{w}_g$. Disso, temos:

$$\frac{\partial}{\partial \mathbf{w}_g}\mathcal{L}(\mathbf{W}) = \begin{bmatrix} \frac{\partial}{\partial w_{g0}}\mathcal{L}(\mathbf{W}) \\ \frac{\partial}{\partial w_{g2}}\mathcal{L}(\mathbf{W}) \\ \vdots \\ \frac{\partial}{\partial w_{gd}}\mathcal{L}(\mathbf{W}) \end{bmatrix}$$

Queremos calcular, então, $\frac{\partial}{\partial w_{gh}}\mathcal{L}(\mathbf{W})$ para $g = 1, \ldots, K$ e $h = 1, \ldots, d$.

$$\frac{\partial}{\partial w_{gh}}\mathcal{L}(\mathbf{W}) = \frac{\partial}{\partial w_{gh}}\left(-\frac{1}{N}\sum_{i=1}^{N}\sum_{q=1}^{K} y_q^{(i)} \ln(\hat{y}_q^{(i)})\right) =$$

$$= -\frac{1}{N}\sum_{i=1}^{N}\sum_{q=1}^{K}\frac{\partial}{\partial w_{gh}}\left(y_q^{(i)} \ln(\hat{y}_q^{(i)})\right) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{q=1}^{K} y_q^{(i)}\frac{\partial}{\partial w_{gh}}\ln(\hat{y}_q^{(i)})$$

Perceba que $\hat{y}_q^{(i)} = \frac{e^{\mathbf{w}_q^T \tilde{\mathbf{x}}^{(\mathbf{i})}}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^T \tilde{\mathbf{x}}^{(\mathbf{i})}}}$ então para qualquer q, g, h, e i, $\hat{y}_q^{(i)}$ depende de w_{gh}. Então,

$$\frac{\partial}{\partial w_{gh}}\ln(\hat{y}_q^{(i)}) = \frac{1}{\hat{y}_q^{(i)}} \cdot \frac{\partial}{\partial w_{gh}}\hat{y}_q^{(i)}$$

Para calcular $\frac{\partial}{\partial w_{gh}}\hat{y}_q^{(i)}$, vamos tratar dois casos:

1. $q = g$

$$\frac{\partial}{\partial w_{gh}}\hat{y}_g^{(i)} = \frac{\partial}{\partial w_{gh}}\left(\frac{e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}}{\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}}\right) =$$

$$= \frac{\frac{\partial}{\partial w_{gh}}\left(e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}\right)\cdot\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}} - e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}\frac{\partial}{\partial w_{gh}}\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}}{\left(\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}\right)^2} =$$

$$= \frac{e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}\cdot\frac{\partial}{\partial w_{gh}}\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}\cdot\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}} - e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}\frac{\partial}{\partial w_{gh}}e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}}{\left(\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}\right)^2} =$$

$$= \frac{e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}\cdot x_h^{(i)}\cdot\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}} - e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}\cdot x_h^{(i)}}{\left(\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}\right)^2} =$$

$$= \frac{e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}}{\left(\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}\right)^2}\cdot x_h^{(i)} - \frac{\left(e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}\right)^2}{\left(\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}\right)^2}\cdot x_h^{(i)} =$$

$$= \hat{y}_g^{(i)}x_h^{(i)} - \hat{y}_g^{(i)^2}x_h^{(i)} = \hat{y}_g^{(i)}x_h^{(i)}\left(1-\hat{y}_g^{(i)}x_h^{(i)}\right)$$

2. $q \neq g$

$$\frac{\partial}{\partial w_{gh}}\hat{y}_q^{(i)} = \frac{\frac{\partial}{\partial w_{gh}}\left(e^{\mathbf{w}_q^T\tilde{\mathbf{x}}^{(i)}}\right)\cdot\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}} - e^{\mathbf{w}_q^T\tilde{\mathbf{x}}^{(i)}}\frac{\partial}{\partial w_{gh}}\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}}{\left(\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}\right)^2} =$$

$$= \frac{-e^{\mathbf{w}_q^T\tilde{\mathbf{x}}^{(i)}}\frac{\partial}{\partial w_{gh}}\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}}{\left(\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}\right)^2} = \frac{-e^{\mathbf{w}_q^T\tilde{\mathbf{x}}^{(i)}}\frac{\partial}{\partial w_{gh}}e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}}{\left(\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}\right)^2} =$$

$$= \frac{-e^{\mathbf{w}_q^T\tilde{\mathbf{x}}^{(i)}}e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}\cdot x_h^{(i)}}{\left(\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}\right)^2} = \frac{-e^{\mathbf{w}_q^T\tilde{\mathbf{x}}^{(i)}}}{\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}}\frac{e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}}{\sum_{j=1}^K e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}}x_h^{(i)} =$$

$$= -\hat{y}_q^{(i)}\hat{y}_g^{(i)}x_h^{(i)}$$

Sumarizando, temos:

$$\frac{\partial}{\partial w_{gh}}\mathcal{L}(\mathbf{W}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_g^{(i)}\frac{1}{\hat{y}_g^{(i)}}\frac{\partial}{\partial w_{gh}}\hat{y}_g^{(i)}\sum_{\substack{q=1\\q\neq g}}^{K} y_q^{(i)}\frac{1}{\hat{y}_q^{(i)}}\frac{\partial}{\partial w_{gh}}\hat{y}_q^{(i)}\right)$$

Aqui, somente abrimos o somatório em dois termos diferentes, tais que do lado esquerdo temos o termo tal que $q = g$ e, do lado direito, $q \neq g$.

$$\frac{\partial}{\partial w_{gh}}\mathcal{L}(\mathbf{W}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_g^{(i)}\frac{1}{\hat{y}_g^{(i)}}\hat{y}_g^{(i)}x_h^{(i)}\left(1-\hat{y}_g^{(i)}x_h^{(i)}\right) + \sum_{\substack{q=1\\q\neq g}}^{K} y_q^{(i)}\frac{1}{\hat{y}_q^{(i)}} - \hat{y}_q^{(i)}\hat{y}_g^{(i)}x_h^{(i)}\right) =$$

$$= \frac{1}{N}\sum_{i=1}^{N}\left(y_g^{(i)}\hat{y}_g^{(i)}x_h^{(i)} - y_g^{(i)}x_h^{(i)} + \sum_{\substack{q=1\\q\neq g}}^{K} y_q^{(i)}\hat{y}_g^{(i)}x_h^{(i)}\right)$$

Agora, podemos agrupar o somatório novamente:

$$\frac{\partial}{\partial w_{gh}}\mathcal{L}(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N}\left(-y_g^{(i)}x_h^{(i)} + \sum_{q=1}^{K} y_q^{(i)}\hat{y}_g^{(i)}x_h^{(i)}\right)$$

Como $y^{(i)}$ é um vetor One-Hot Encoded:

$$\frac{\partial}{\partial w_{gh}}\mathcal{L}(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N}\left(-y_g^{(i)}x_h^{(i)} + \hat{y}_g^{(i)}x_h^{(i)}\right) = \frac{1}{N}\sum_{i=1}^{N} x_h^{(i)}\left(\hat{y}_g^{(i)} - y_g^{(i)}\right)$$

Generalizando esse valor para $h = 1, \ldots, d$ encontramos $\dfrac{\partial}{\partial \mathbf{w}_g}\mathcal{L}(\mathbf{W})$. De maneira análoga, se generalizarmos tal achado para $g = 1, \ldots, K$, teremos $\nabla\mathcal{L}(\mathbf{W})$.

5.3.3.2 Gradiente Descendente

Portanto, podemos explicitar o passo de atualização do vetor de pesos $\mathbf{W}$ da Regressão Logística Multinomial:

$$w_{gh}(t+1) \leftarrow w_{gh}(t) - \eta\left(\frac{1}{N}\sum_{i=1}^{N} x_h^{(i)}\left(\frac{e^{\mathbf{w}_g^T\tilde{\mathbf{x}}^{(i)}}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^T\tilde{\mathbf{x}}^{(i)}}} - y_g^{(i)}\right)\right)$$

5.4 Aplicação

Como foi exposto, é possível fazer um modelo de Regressão Multinomial utilizando a função Softmax, utilizando a função de perda, e gradiente desenvolvidos ao longo dessa seção. Tal implementação segue o mesmo padrão das implementações de Regressão Logística e Linear. Além disso, como será mencionado mais à frente, a função Softmax é comumente utilizada como função de ativação da última camada de Redes Neurais para classificação multinomial. Nesse sentido, evitando redundâncias, realizei somente a implementação que conjuga Softmax com Redes Neurais e tal trabalho está presente na última seção deste trabalho.

6 Redes Neurais

6.1 Perceptron

Para contextualizar o estudo de Redes Neurais, é interessante introduzirmos o conceito do Perceptron. Recuperando o que já foi exposto sobre o Modelo Linear, podemos expressar o Perceptron como um classificador binário, utilizado quando as classes em questão são linearmente separáveis. O classificador pode ser expressado pelo que segue:

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^T\tilde{\mathbf{x}})$$

Perceba que, enquanto temos exemplos $(\mathbf{x}_i, y_i)$ $i = 1, \ldots, N$ tais que $\mathbf{x}_i \in \mathbb{R}^d$ e $y_i \in \{-1, +1\}$, o Perceptron é uma função $h : \mathbb{R}^d \to \{-1, +1\}$.

De maneira geral, podemos interpretar o modelo linear $\mathbf{w}^T\tilde{\mathbf{x}}$ como uma função linear cujos pontos tais que $\mathbf{w}^T\tilde{\mathbf{x}} = 0$ formam um hiperplano em $\mathbb{R}^d$ que será separador dos dados. Assim, quando $\mathbf{w}^T\tilde{\mathbf{x}} > 0$ queremos nos referir à uma classe de dados e quando $\mathbf{w}^T\tilde{\mathbf{x}} < 0$ nos referimos à outra.

Considere o seguinte exemplo com $d = 1$:

Exemplo 6.1 *Se quisermos ilustrar um Perceptron tal que* $h(\mathbf{x}) = sign(\mathbf{w}^T\tilde{\mathbf{x}})$, *onde* $\mathbf{x} \in \mathbb{R}$, *poderemos considerar pares de exemplos do tipo:*

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$$

E a função que separa tais dados, então, deve ser do tipo:

$$h(x_i) = sign\left(\mathbf{w}^T\tilde{\mathbf{x}}_i\right) = sign\left(\begin{bmatrix} w_0 \\ w_1 \end{bmatrix}^T \begin{bmatrix} 1 \\ x_i \end{bmatrix}\right) = sign\left(w_0 + w_1 x_i\right)$$

Então, podemos considerar os seguintes dados nesse exemplo:

$$A, B = (x_1, y_1), (x_2, y_2) = (3, 1), (1, -1)$$

Como os dados de entrada possuem dimensão igual a 1*, eles estão simplesmente dispostos na reta. Por isso, o ponto que pertence à classe positiva (A) será pintado de azul e o ponto que pertence à classe negativa (B) será pintado de vermelho.*

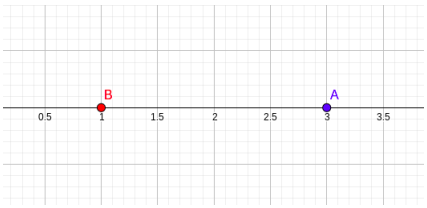

Figura 4: Pontos A e B

A expressão para esse Perceptron pode ser $h(x_i) = sign\,(-4 + 2x_i)$*. Perceba que* $\mathbf{w}^T\tilde{\mathbf{x}} = -4 + 2x$ *é uma função linear que respeita a configuração desses pontos, pois*

$$sign\,(-4 + 2(3)) = +1 \;\; e \;\; sign\,(-4 + 2(1)) = -1$$

Portanto, a expressão $-4 + 2x = 0 \Rightarrow x = 2$ *resulta em um ponto (hiperplano em* $\mathbb{R}$*) que é separador das classes em questão. Observe na figura 5.*

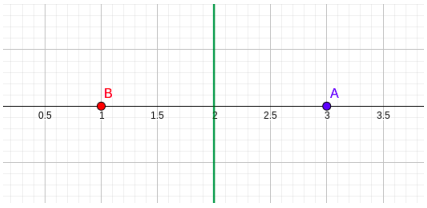

Figura 5: Hiperplano separador

Assim, nesse exemplo, quando quisermos classificar novos dados utilizando esse Perceptron obtido, teremos que:

$$h(x) = sign\,(-4 + 2x)$$

6.1.1 PLA - Algoritmo de Aprendizado Perceptron

Sabendo da tarefa do Perceptron, de selecionar um hiperplano de dimensão $d+1$ para separar linearmente os exemplos fornecidos:

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$$

como obter — aprender — os pesos do vetor $\mathbf{w}$ de modo que $h(\mathbf{x}) = \text{sign}(\mathbf{w}^T\tilde{\mathbf{x}})$ alcance esse objetivo?

Nesse ponto, entra o algoritmo de aprendizado Perceptron, que é curiosamente simples e o qual (sob algumas premissas) conseguimos provar que converge para um hiperplano que separe os exemplos linearmente.

Esse algoritmo foi um dos pioneiros para classificação binária e funciona da seguinte maneira: Primeiro incializamos o vetor pesos $\mathbf{w}$ com valores aleatórios — todo o processo consistirá em fazer ajustes à esse vetor de pesos para que ele seja capaz de separar os pares de dados em questão.

Nesse sentido, enquanto existir um par de dados $(\mathbf{x}_i, y_i)$ no conjunto de treinamento tal que $\text{sign}(\mathbf{w}^T\tilde{\mathbf{x}}_i) \neq y_i$, atualizaremos o vetor de pesos da seguinte maneira:

$$\mathbf{w} \leftarrow \mathbf{w} + y_i\tilde{\mathbf{x}}_i$$

Dado que os tais pontos são linearmente separáveis, o PLA eventualmente virá a convergir. Então, nesse momento, não haverá mais pontos i tais que $\text{sign}(\mathbf{w}^T\tilde{\mathbf{x}}_i) \neq y_i$. Isto é, o vetor de pesos $\mathbf{w}$ encontrado classifica todos os dados de treinamento corretamente. Por isso, esse vetor é a resposta procurada.

Se quisermos entender mais a fundo o motivo desse algoritmo tão simples funcionar, é possível obter uma noção gráfica a partir do passo de atualização do vetor de pesos. Podemos, ainda, encontrar — no seguinte link Convergência do PLA — a prova de convergência desse algoritmo.

Como o Perceptron funciona exclusivamente para problemas linearmente separáveis, esse modelo de Machine Learning serve, atualmente, principalmente à fins didáticos. Assim, podemos não nos ater às suas particularidades e, por conseguinte, dar prosseguimento à esse estudo.

6.1.2 Perceptron como abstração de um neurônio

No contexto de Redes Neurais, podemos interpretar Perceptrons como uma espécie de neurônio artificial — cuja idealização e formulação é, de

certo modo, inspirada em neurônios biológicos. Nesse sentido, um Perceptron é constituído de entradas $x_i,\ i = 0, \ldots, d$ que são ponderadas por pesos $w_i,\ i = 0, \ldots, d$ e, disso, apresenta uma única saída binária $\text{sign}(\mathbf{w}^T\tilde{\mathbf{x}})$.

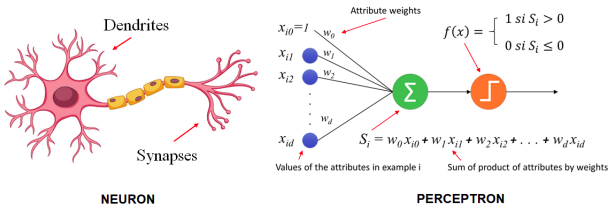

Figura 6: Analogia de um Perceptron como um neurônio biológico

6.2 Multilayer Perceptron - MLP

Com a explicação sobre o Perceptron, podemos perceber que, a depender de dados de entrada, utilizando essa abordagem, conseguimos — com base na entrada — gerar decisões binárias.

6.2.1 Funções Lógicas

A ideia de "Perceptrons Multicamadas" é combinar diversos Perceptrons para que possamos tomar decisões mais sutis. Desse modo, apesar de um Perceptron aceitar um vetor de números reais $\mathbf{x}$ como entrada, para motivar o estudo de MLP's é comum utilizarmos Perceptrons para implementarmos funções lógicas.

Para funções mais simples, como **AND**, **OR**, que constituem problemas linearmente separáveis, podemos usar somente um Perceptron para formular essas portas lógicas, mas quando desejamos funções mais complicadas, como **XOR**, devemos utilizar combinações de Perceptrons, e é aí que o MLP entra em cena.

6.2.1.1 Função Degrau

Para essa aplicação, faremos uma pequena mudança em nossa definição de Perceptron. Como funções lógicas são binárias com saída $\in \{0, 1\}$, definiremos o Perceptron da seguinte maneira:

$$g(\mathbf{x}) = \begin{cases} 0 \ se \ \mathbf{w}^T\tilde{\mathbf{x}} \leq 0 \\ 1 \ se \ \mathbf{w}^T\tilde{\mathbf{x}} > 0 \end{cases}$$

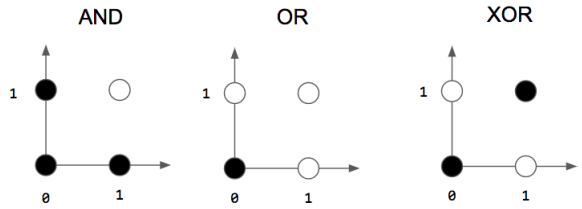

Figura 7: Representação gráfica das funções lógicas **AND**, **OR** e **XOR**

Chamaremos essa função degrau de função de ativação de um neurônio Perceptron.

Perceba que, as próximas expressões obtidas para Perceptrons que reproduzem o comportamento de portas lógicas são, na prática os hiperplanos separadores de classes. Nesse caso, retas em $\mathbb{R}^2$. Isso por que cada ponto i de exemplo é tal que $\mathbf{x}_i \in \{\{0,1\} \times \{0,1\}\}$.

6.2.1.2 AND

Como visto na figura 7, a resposta esperada de uma porta **AND** é constituída por pontos linearmente separáveis. Assim, a porta **AND** pode ser representada pelo Perceptron:

$$g(x_1, x_2) = x_1 + x_2 - \frac{3}{2}$$

6.2.1.3 OR

De modo análogo ao anterior, o Perceptron que pode ser utilizado como a porta **OR** é o seguinte:

$$g(x_1, x_2) = x_1 + x_2 - \frac{1}{2}$$

6.2.1.4 NAND

Sobre a função lógica **NAND**, que é o complementar da função **AND** e, por isso, também possui dados linearmente separáveis, podemos utilizar um Perceptron para implementá-la do seguinte modo:

$$g(x_1, x_2) = -x_1 - x_2 + \frac{3}{2}$$

Sobre essa função, é interessante observarmos que ela é uma função lógica universal. Isso quer dizer que poderemos formar qualquer outra função lógica

utilizando combinações de **NAND**'s. Para ilustrar isso, veremos o caso da porta **XOR** (que têm dados não linearmente separáveis e motiva a utilização de MLP).

6.2.1.5 XOR

Como conseguimos ver na figura 7, no caso da **XOR**, não é possível utilizar somente um Perceptron para reproduzir o comportamento dessa função lógica. Agora, podemos combinar portas **NAND** como é mostrado na figura 8 para alcançar o resultado da **XOR**.

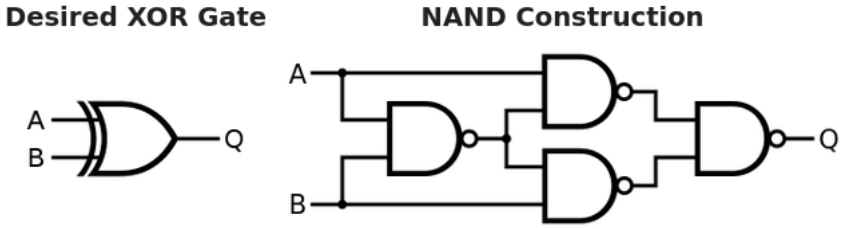

Figura 8: Construção da porta lógica XOR a partir da NAND.

É interessante notar que a combinação de Perceptrons em camadas, evidente nesse exemplo, demonstra a habilidade de solucionar problemas não linearmente separáveis e isso é motivador para estudar Redes Neurais.

Relembrando a ideia principal desse, texto, que se baseia no aprendizado baseado em dados, a essência da utilização de técnicas de Machine Learning é aproximar uma função alvo (f) por h. Nesse sentido, até agora, trabalhamos com funções h que, por essência, têm um comportamento baseado na linearidade.

Dessa maneira, como bem exposto pelo caso da porta lógica **XOR**, em grande parte das vezes, não trabalharemos com funções com comportamento que pode ser modelado utilizando tais estratégias. Contudo, seguindo um caminho parecido com o da porta lógica **NAND**, a combinação dessas técnicas baseadas em linearidade pode resultar em estruturas universais, que sejam capazes de produzir h que aproxime absolutamente qualquer f.

Essas estruturas são as Redes Neurais. Assim, nos baseando — mais precisamente — em unidades sigmoides (que serão o tópico da próxima seção), podemos produzir Redes que são capazes de aproximar qualquer função alvo

f. Essa afirmação é conhecida como Teorema da aproximação universal (*Universal approximation theorem*) e pode ser provada matematicamente. Dessa forma, podemos atribuir à esse fato o sucesso e ampla utilização de Redes Neurais no contexto Machine Learning e inteligência Artificial nos dias atuais. Portanto, seguimos esse estudo abordando esse tópico tão importante.

6.3 Neurônio com ativação sigmoide

Quando tentando resolver problemas mais complicados utilizando MLP, como estamos trabalhando como classificadores binários, é comum que pequenas variações em alguma parte da rede, mude a saída de um nó de 0 para 1 e isso faz com que o treinamento da rede se torne excessivamente complicado. Para atacar esse problema, mudamos a função de ativação. Deixamos de utilizar a função degrau — com saída binária — e passamos a utilizar uma função Sigmoide — com saída contínua entre 0 e 1.

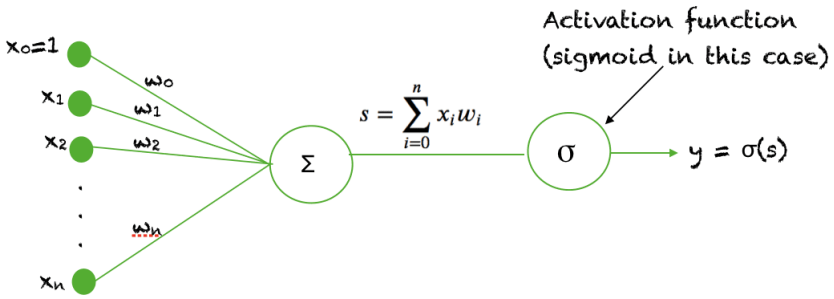

Figura 9: Representação gráfica de um neurônio sigmoide

Nesse sentido, é interessante notar que o algoritmo de Regressão Logística, como discutido em seções anteriores, pode ser enxergado como um neurônio. Contudo, podemos também — de maneira mais relaxada — definir um neurônio sigmoide como um neurônio que implementa uma função degrau "suavizada" e contínua.

Desse modo, apesar de existirem outras funções de ativação que seguem o padrão descrito, como a de tanh (tangente hiperbólica) e ReLU (*Rectified Linear Unity*), é muito comum que a função de ativação dos neurônios de redes neurais seja a que deduzimos durante o estudo de Regressão Logística. Portanto, basearemos bastante o estudo de Redes Neurais nessa unidade importante.

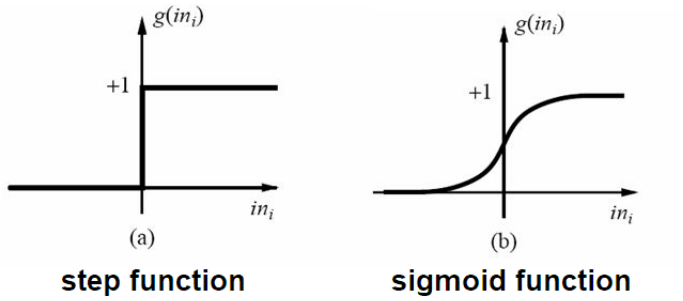

Figura 10: Comparação entre as funções Sigmoide e Degrau

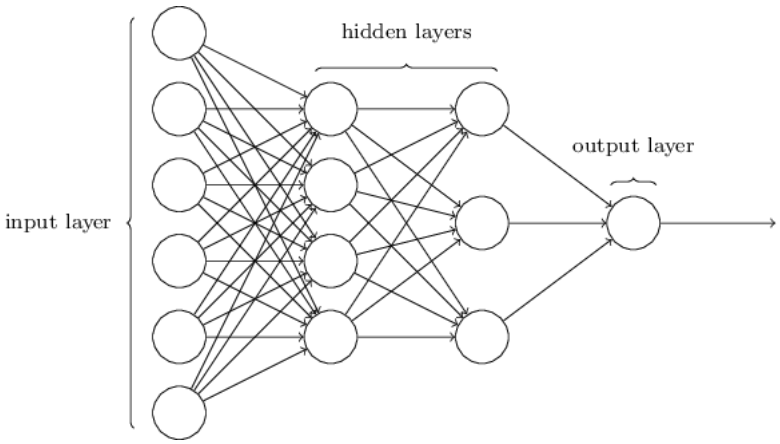

Figura 11: Estrutura de uma Rede Neural

6.4 Estrutura de uma Rede Neural

Seguindo a ideia de combinar múltiplos neurônios para resolver problemas mais complexos, é válido examinarmos como é a arquitetura de uma Rede Neural composta de neurônios sigmoides.

Quando representamos os neurônios enfileirados, formando colunas, estamos segmentando a rede em camadas e quando utilizamos diversas camadas, dizemos que se trata de uma rede *multi-layer*. O formato padrão — e mais elementar — de redes neurais podem ser caracterizadas como *feedfoward*.

Esse tipo de rede não possui loops de retorno (*feedback*) e é *fully connected*. Ou seja, cada neurônio, de cada camada, somente possui conexões com neurônios da próxima camada. Ainda, o fato de ser *fully connected* indica que cada neurônio será conectado à todos os neurônio da camada seguinte.

Dessa forma, as camadas de uma rede podem ser definidas das seguintes

maneiras:

- **De entrada**
  Nessa camada, há a entrada dos dados, representados pelo vetor $\mathbf{x}$.
- **Ocultas**
  Camadas que ficam entre a de entrada e de saída.
- **De saída**
  É daqui que tiramos o valor da previsão $\hat{y}$.

Nessa forma de representação, as arestas que conectam um nó (neurônio) a outro são os pesos que compõem o vetor de pesos $\mathbf{w}$. Assim, cada nó é alimentado pelo somatório dos valores dos nós da camada anterior, ponderados por pesos. Ou seja, cada nó é alimentado pela já familiar expressão $\mathbf{w}^T\tilde{\mathbf{x}}$ e assume valor $\sigma\left(\mathbf{w}^T\tilde{\mathbf{x}}\right)$. Essa passagem de valores de camada a camada, até a camada de saída, é chamada de "forward pass".

6.5 Notação

Utilizaremos a seguinte representação gráfica para auxiliar a definição da notação que utilizaremos para tratar de redes neurais.

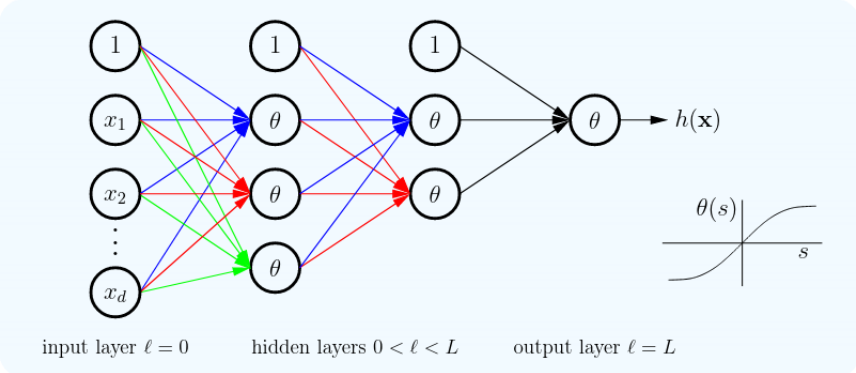

Figura 12: Exemplo de Rede Neural

Em uma rede neural genérica, teremos as camadas $l = 0, 1, \ldots, L$. Para indicar cada camada, usaremos o índice$^{(l)}$, sobrescrito. Desse modo, cada camada terá "dimensão" $d^{(l)}$, ou seja, terá $d^{(l)} + 1$ nós. Por sua vez, os nós serão referenciados por $0, 1, \ldots, d^{(l)}$. Perceba que o nó 0 cumpre papel do "bias" em nossa rede — o elemento $x_0 = 1$ inserido no vetor de elementos $\mathbf{x}$ para compor o modelo linear.

Para continuar a tarefa de definição de notação, analisaremos em detalhe a relação entre dois nós da rede, como mostra a figura 13.

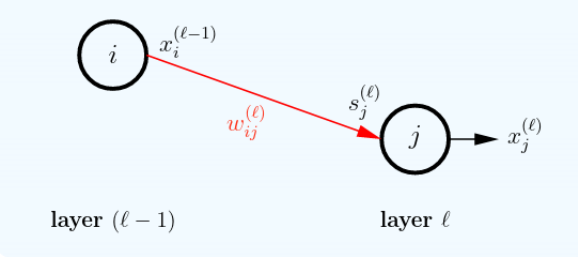

Figura 13: Relação entre dois nós

Perceba que os nós possuem sinais de entrada s e de saída x. Contudo, os nós na camada de entrada ($l = 0$) dão à rede os valores de entrada $x_1, \ldots, x_d$ e, por isso, não têm sinais de entrada. Além disso, os nós 0 de cada camada também não possuem sinais de entrada e têm valor de saída igual à 1.

No caminho de incorporar esses elementos em uma notação matricial, olharemos para a rede camada à camada. Assim, como temos sinais de entrada nos nós $1, \ldots, d^{(l)}$ da camada l, teremos o vetor $\mathbf{s}^{(l)}$. Por outro lado, como, nessa mesma camada, temos sinais de saída nos nós $0, 1, \ldots, d^{(l)}$, teremos o vetor $\mathbf{x}^{(l)}$. Perceba que $\mathbf{x}^{(l)}$ possui $d^{(l)} + 1$ elementos enquanto $\mathbf{s}^{(l)}$ possui $d^{(l)}$.

Quanto aos pesos, note que, na camada l, cada nó recebe entradas de todos os $d^{(l-1)+1}$ nós da camada anterior. Por tanto, teremos uma matriz de pesos $\mathrm{W}^{(l)}$ com tamanho $d^{(l-1)} + 1 \times d^{(l)}$, onde o elemento $w_{ij}^{(l)}$ é o peso que une o nó i da camada $l - 1$ ao nó j da camada l.

Portanto, o conjunto das matrizes $\mathrm{W}^{(l)}$ $l = 1, \ldots, L$ equivale à todos os pesos presentes na rede neural em questão e constituem o parâmetro de pesos $\mathbf{w} = \{\mathrm{W}^{(1)}, \mathrm{W}^{(2)}, \ldots, \mathrm{W}^{(L)}\}$.

6.6 Forward Propagation

Agora, a partir da representação gráfica de uma rede neural e da exposição dos vetores de sinais de entrada $\mathbf{s}^{(l)}$, de saída $\mathbf{x}^{(l)}$ e de pesos $\mathrm{W}^{(l)}$, queremos chegar à expressões (matriciais) que representem a passagem dos inputs $x_1, \ldots, x_d$ até a camada final, de modo a obter o resultado de $h(\mathbf{w})$.

Nesse sentido, note que podemos relacionar as entradas e saídas de uma camada da rede neural da seguinte forma:

$$\mathbf{x}^{(l)} = \begin{bmatrix} 1 \\ \sigma(\mathbf{s}^{(l)}) \end{bmatrix}$$

Ainda, note que, sobre o sinal de entrada no nó j da camada l,

$$s_j^{(l)} = \sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}$$

Examinando a fundo tal igualdade, temos o que segue:

$$\mathrm{W}^{(l)} = \begin{bmatrix} w_{0\,1}^{(l)} & w_{0\,2}^{(l)} & \dots & w_{0\,d^{(l)}} \\ w_{1\,1}^{(l)} & w_{1\,2}^{(l)} & \dots & w_{1\,d^{(l)}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d^{(l-1)}\,1}^{(l)} & w_{d^{(l-1)}\,2}^{(l)} & \dots & w_{d^{(l-1)}\,d^{(l)}} \end{bmatrix}_{d^{(l-1)} \times d^{(l)}} \qquad \mathbf{x}^{(l-1)} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{d^{(l-1)}} \end{bmatrix}_{d^{(l-1)} \times 1}$$

Note que $\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}$ emparelha a coluna j de $\mathrm{W}^{(l)}$ com a única coluna do vetor $\mathbf{x}^{(l)}$. Se tal emparelhamento ocorresse com a linha j de $\mathrm{W}^{(l)}$, teríamos um produto entre matriz e vetor. Dessa forma, podemos transpor a matriz $\mathrm{W}^{(l)}$ para obter esse resultado:

$$\left(\mathrm{W}^{(l)}\right)^T = \begin{bmatrix} w_{0\,1}^{(l)} & w_{1\,1}^{(l)} & \dots & w_{d^{(l-1)}\,1}^{(l)} \\ w_{0\,2}^{(l)} & w_{1\,2}^{(l)} & \dots & w_{d^{(l-1)}\,2}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{0\,d^{(l)}} & w_{1\,d^{(l)}} & \dots & w_{d^{(l-1)}\,d^{(l)}} \end{bmatrix}_{d^{(l)} \times d^{(l-1)}} \qquad \mathbf{x}^{(l-1)} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{d^{(l-1)}} \end{bmatrix}_{d^{(l-1)} \times 1}$$

Disso, conseguimos a expressão:

$$\mathbf{s}^{(l)} = \left(\mathrm{W}^{(l)}\right)^T \mathbf{x}^{(l-1)} \tag{1}$$

Assim, para obter $h(\mathbf{x}) = \hat{y}$ a partir do vetor de entrada, basta efetuar os seguintes passos:

$$\mathbf{x} = \mathbf{x}^{(0)} \xrightarrow{\mathrm{W}^{(1)}} \mathbf{s}^{(1)} \xrightarrow{\sigma} \mathbf{x}^{(1)} \xrightarrow{\mathrm{W}^{(2)}} \mathbf{s}^{(2)} \xrightarrow{\sigma} \mathbf{x}^{(2)} \cdots \to \mathbf{s}^{(L)} \xrightarrow{\sigma} \mathbf{x}^{(L)} = h(\mathbf{x})$$

6.7 Treinamento de uma Rede Neural

Conseguimos definir a maneira como os dados de entrada $x_1, \ldots, x_d$ se propagam da entrada da rede neural até sua saída, produzindo seu resultado. Mas, a essência de uma rede neural que funcione adequadamente é possuir o parâmetro de pesos $\mathbf{w} = \{\mathrm{W}^{(1)}, \mathrm{W}^{(2)}, \ldots, \mathrm{W}^{(L)}\}$ ajustado de forma que o resultado $h(\mathbf{x})$ seja o mais próximo do da função alvo f que queremos aproximar. Assim, descreveremos a seguir como o processo de treinamento de uma rede neural ocorre.

Primeiramente, precisamos definir uma função de custo. Isso é feito, principalmente, nos baseando no funcionamento e função de ativação da última camada da rede. Com isso, otimizaremos essa função para obter os pesos que minimizam a função de custo/perda.

Contudo, para que o processo de otimização se dê de maneira eficiente, utilizaremos o método numérico do gradiente descendente estocástico e, para calcular os gradientes envolvidos nessa técnica, vamos empregar um algoritmo chamado *Backpropagation*.

6.7.1 Backpropagation

O algoritmo de Backpropagation, é uma maneira eficiente de encontrarmos o gradiente $\nabla\mathcal{L}(\mathbf{w})$ de uma função de erro da nossa rede neural em relação a seus pesos. Primeiramente, precisamos considerar que o erro medido por todo o conjunto de treino pode ser representado como a média do erro obtido em cada exemplo de treinamento:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \mathrm{erro}_i$$

Por isso, ao longo desse texto, mantivemos a prática de fazer com que as funções de perda deduzidas para cada modelo estudado respeitasse esse padrão. Note que $\mathrm{erro}_i = \mathrm{e}\left(\hat{y}_i, y_i\right)$. Para obter esse gradiente, queremos:

$$\frac{\partial\mathcal{L}(\mathbf{w})}{\partial\mathrm{W}^{(l)}} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial\mathrm{erro}_i}{\partial\mathrm{W}^{(l)}}$$

Então, o objetivo desse algoritmo é justamente obter tais derivadas parciais em relação às matrizes de pesos. A ideia aqui é obter as derivadas parciais de uma camada l a partir das derivadas parciais da camada $l+1$.

Para fazer isso, deduziremos duas equações importantes que nos ajudarão a encontrar o gradiente procurado, as derivadas parciais do erro em função das matrizes de peso (que estará em função dos vetores de sensibilidades) e os vetores de sensibilidades de cada camada.

6.7.1.1 Derivada parcial do erro em função das matrizes de peso

Primeiramente, definimos um vetor de sensibilidades:

$$\boldsymbol{\delta}^{(l)} = \frac{\partial \text{erro}}{\partial \mathbf{s}^{(l)}} = \begin{bmatrix} \dfrac{\partial \text{erro}}{\partial \mathbf{s}_1^{(l)}} \\ \dfrac{\partial \text{erro}}{\partial \mathbf{s}_2^{(l)}} \\ \vdots \\ \dfrac{\partial \text{erro}}{\partial \mathbf{s}_{d^{(l)}}^{(l)}} \end{bmatrix}$$

Então, temos que:

$$\frac{\partial \text{erro}}{\partial \mathrm{W}^{(l)}} = \mathbf{x}^{(l-1)} \left(\boldsymbol{\delta}^{(l)}\right)^T \tag{2}$$

Para entender como tal expressão se origina, vamos examinar a sequência de dependências já estabelecida:

$$\mathbf{x} = \mathbf{x}^{(0)} \xrightarrow{\mathrm{W}^{(1)}} \mathbf{s}^{(1)} \xrightarrow{\sigma} \mathbf{x}^{(1)} \xrightarrow{\mathrm{W}^{(2)}} \mathbf{s}^{(2)} \xrightarrow{\sigma} \mathbf{x}^{(2)} \cdots \to \mathbf{s}^{(L)} \xrightarrow{\sigma} \mathbf{x}^{(L)} = h(\mathbf{x})$$

Desse modo, perceba que, se tratando de qualquer camada (l), o erro depende de $\mathbf{s}^{(l)}$ que, por sua vez, depende de $\mathrm{W}^{(l)}$. Então, se efetuarmos essa derivação a partir dos elementos de cada matriz $\mathrm{W}^{(l)}$, podemos usar a regra da cadeia da seguinte maneira:

$$\frac{\partial \text{erro}}{\partial w_{ij}^{(l)}} = \frac{\partial \text{erro}}{\partial \mathbf{s}_j^{(l)}} \cdot \frac{\partial \mathbf{s}_j^{(l)}}{\partial w_{ij}^{(l)}}$$

Agora, observando a expressão (1), conseguimos notar que o elemento $\mathbf{s}_j^{(l)}$ advém da linha j de $\left(\mathrm{W}^{(l)}\right)^T$, isto é, da coluna j de $\mathrm{W}^{(l)}$ e do vetor coluna $\mathbf{x}^{(l-1)}$. Ou seja:

$$\mathbf{s}_j^{(l)} = \sum_{n=1}^{d^{(l-1)}} w_{nj}^{(l)} \mathbf{x}_n^{(l-1)} \tag{3}$$

Então, temos:

$$\frac{\partial \mathbf{s}_j^{(l)}}{\partial w_{ij}^{(l)}} = \frac{\partial}{\partial w_{ij}^{(l)}} w_{ij}^{(l)} \mathbf{x}_i^{(l-1)} = \mathbf{x}_i^{(l-1)}$$

Lembrando da definição do vetor de sensibilidades, $\frac{\partial \text{erro}}{\partial \mathbf{s}_j^{(l)}} = \boldsymbol{\delta}_j^{(l)}$, então:

$$\frac{\partial \text{erro}}{\partial w_{ij}^{(l)}} = \mathbf{x}_i^{(l-1)} \cdot \boldsymbol{\delta}_j^{(l)}$$

E essa é a representação de (2) a partir de seus componentes.

6.7.1.2 Vetor de sensibilidades em cada camada

A fórmula para os vetores de sensibilidades que iremos utilizar no algoritmo de backpropagation é

$$\boldsymbol{\delta}^{(l)} = \sigma'(\mathbf{s}^{(l)}) \otimes \left[\mathrm{W}^{(l+1)} \boldsymbol{\delta}^{(l+1)}\right]_1^{d^{(l)}} \tag{4}$$

Onde σ' é a derivada da função de ativação da camada (l) e o vetor $\left[\mathrm{W}^{(l+1)} \boldsymbol{\delta}^{(l+1)}\right]_1^{d^{(l)}}$ é o vetor $\mathrm{W}^{(l+1)} \boldsymbol{\delta}^{(l+1)}$ excluindo o seu primeiro elemento, bias, que possui índice 0. Além disso, o símbolo $\otimes$ denota o produto de Hadamard, ou seja, produto componente a componente dos vetores em questão.

Para deduzir essa expressão, primeiramente devemos pensar na expressão que definimos para mensurar o erro da Rede Neural.

$$\text{erro} = \text{erro}\,(\hat{y}, y) = \text{erro}\left(\mathbf{x}^{(L)}, y\right) = \text{erro}\left(\sigma(\mathbf{s}^{(L)}), y\right) \tag{5}$$

Disso, conseguimos perceber que o erro depende de $\mathbf{s}^{(L)}$ exclusivamente a partir de $\mathbf{x}^{(L)}$. Então, se derivarmos o erro em relação a $\mathbf{s}^{(l)}$, pela regra da cadeia, temos que:

$$\frac{\partial \text{erro}}{\partial \mathbf{s}^{(l)}} = \frac{\partial \text{erro}}{\partial \mathbf{x}^{(l)}} \cdot \frac{\partial \mathbf{x}^{(l)}}{\partial \mathbf{s}^{(l)}}$$

Portanto, voltando ao vetor de sensibilidades (sobre seus componentes):

$$\boldsymbol{\delta}_j^{(l)} = \frac{\partial \text{erro}}{\partial \mathbf{s}_j^{(l)}} = \frac{\partial \text{erro}}{\partial \mathbf{x}_j^{(l)}} \cdot \frac{\partial \mathbf{x}_j^{(l)}}{\partial \mathbf{s}_j^{(l)}}$$

Agora, usamos o fato de $\mathbf{x}^{(l)}$ ser uma função de $\mathbf{s}^{(l)}$:

$$\frac{\partial \mathbf{x}_j^{(l)}}{\partial \mathbf{s}_j^{(l)}} = \sigma'\left(\mathbf{s}^{(l)}\right) \Rightarrow \boldsymbol{\delta}_j^{(l)} = \frac{\partial \text{erro}}{\partial \mathbf{x}_j^{(l)}} \cdot \sigma'\left(\mathbf{s}^{(l)}\right)$$

Nesse momento, resta definirmos $\dfrac{\partial \text{erro}}{\partial \mathbf{x}_j^{(l)}}$. Mais uma vez, observando a expressão (1), é possível notar que $\mathbf{x}^{(l)}$ afeta $\mathbf{s}^{(l+1)}$. Mas, melhor ainda, é observar (3), já que estamos derivando o erro em relação ao componente j de $\mathbf{x}^{(l)}$. Disso, percebe-se que cada componente de $\mathbf{x}^{(l)}$ influencia todos os componentes de $\mathbf{s}^{(l+1)}$. Então, para obter essa derivação, precisamos somar todas essas "influências":

$$\frac{\partial \text{erro}}{\partial \mathbf{x}_j^{(l)}} = \sum_{k=1}^{d^{(l+1)}} \frac{\partial \text{erro}}{\partial \mathbf{s}_k^{(l+1)}} \cdot \frac{\partial \mathbf{s}_k^{(l+1)}}{\partial \mathbf{x}_j^{(l)}} = \sum_{k=1}^{d^{(l+1)}} \boldsymbol{\delta}_k^{(l+1)} \cdot w_{jk}^{(l+1)}$$

Unindo os resultados obtidos temos, justamente, os componentes de (4):

$$\boldsymbol{\delta}_j^{(l)} = \sigma'\left(\mathbf{s}^{(l)}\right) \cdot \sum_{k=1}^{d^{(l+1)}} \boldsymbol{\delta}_k^{(l+1)} \cdot w_{jk}^{(l+1)}$$

6.7.2 Gradiente Descendente

Sumarizando o processo de backpropagation, a atualização dos parâmetros do modelo é da seguinte forma:

$$\mathrm{W}^{(l)} \leftarrow \mathrm{W}^{(l)} - \eta \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathrm{W}^{(l)}}$$

E realizamos as seguintes tarefas para obter $\dfrac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathrm{W}^{(l)}}$:

1. A cada exemplo no conjunto de treinamento, guardamos o valor de $\frac{\partial \text{erro}_i}{\partial \mathrm{W}^{(l)}}$ para produzirmos $\frac{1}{N}\sum_{i=1}^{N} \frac{\partial \text{erro}_i}{\partial \mathrm{W}^{(l)}}$.
2. Onde $\frac{\partial \text{erro}_i}{\partial \mathrm{W}^{(l)}} = \mathbf{x}^{(l-1)}\left(\boldsymbol{\delta}^{(l)}\right)^T$
3. Enquanto $\mathbf{x}^{(l-1)}$ é advindo do processo de Forward Propagation e $\boldsymbol{\delta}^{(l)} = \sigma'(\mathbf{s}^{(l)}) \otimes \left[\mathrm{W}^{(l+1)}\boldsymbol{\delta}^{(l+1)}\right]_1^{d^{(l)}}$
   ($\mathbf{s}^{(l)}$ também vêm da Forward Propagation)

Desse modo, por causa do vetor de sensibilidades, precisamos iniciar tal processo com $(l) = (L)$ e, de (5):

$$\boldsymbol{\delta}^{(L)} = \frac{\partial \text{erro}}{\partial \mathbf{s}^{(L)}} = \frac{\partial}{\partial \mathbf{s}^{(L)}} \text{erro}\left(\mathbf{x}^{(L)}, y\right)$$

Em que $\dfrac{\partial}{\partial \mathbf{s}^{(L)}} \text{erro}\left(\mathbf{x}^{(L)}, y\right)$ será definido exclusivamente pela função de erro escolhida para a Rede Neural. A obtenção desse resultado, de forma prática (para uma função de erro específica) é demonstrada na implementação realizada.

6.7.3 Implementação

Antes de demonstrar uma aplicação de Redes Neurais, é interessante ressaltar que o método que utilizamos até aqui para se calcular o gradiente da função de perda (gradiente descendente) possui uma propriedade interessante, que nos permite empregar variações mais eficientes desse método. Em particular, usaremos o Gradiente Descendente *Mini batch*, que herda heurísticas do Gradiente Descendente Estocástico.

6.7.3.1 Gradiente Descendente *Mini batch*

No caminho de introduzir essa variação do método de otimização do Gradiente Descendente, é válido relembrarmos que todas as funções de erro estudadas até agora — passando por Regressão Linear até Redes Neurais — foram definidas como a média entre o erro de cada par de dados $(\mathbf{x}_i, y_i)$, de acordo com o seguinte padrão:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \text{e}(\hat{y}_i, y_i)$$

Assim, perceba que, cada vez que atualizamos os parâmetros do modelo de Machine Learning que está sendo implementado precisamos passar por todos os N exemplos do conjunto de treinamento. Agora, perceba que se N é grande, esse processo começa a se tornar demorado. Então, a ideia dessas variações é a atualização mais frequente desses parâmetros visando um processo de treinamento mais rápido.

Desse modo, vamos conceituar algumas terminologias comumente usadas:

- Época (*Epoch*):
  Quando, no processo de treinamento de um modelo, consideramos todo o seu conjunto de treinamento.

- *Batch*:
  Batch possui a tradução literal de lote, e é o conjunto de dados considerado a cada atualização dos parâmetros do modelo que estamos treinando.

  Por isso, o método de Gradiente Descendente que empregamos até agora é chamado "**Gradiente Descendente *Batch***", pois considera todo o conjunto de treinamento como um único *batch* e, assim, a cada época, atualizamos o modelo uma única vez.

Voltando para a definição de métodos de otimização mais eficientes, o chamado **Gradiente Descendente Estocástico**, diferentemente do Gradiente Descendente *Batch*, considera somente um exemplo para atualizar os parâmetros do modelo em questão. Ou seja, na prática, temos um *batch* de tamanho 1 e, a cada época, atualizamos os parâmetros N vezes.

Podemos justificar o porquê dessa versão Estocástica do método de otimização funcionar a partir da seguinte propriedade:

A "direção média" do gradiente quando consideramos um único exemplo é equivalente à direção do gradiente quando consideramos todos os exemplos.

Observe:

$$E\left[\nabla \mathrm{e}(\hat{y}_i, y_i)\right] = \frac{1}{N}\sum_{i=1}^{N} \nabla \mathrm{e}(\hat{y}_i, y_i)$$

Isso, porque cada exemplo tem probabilidade $\frac{1}{N}$ de ser escolhido e há N exemplos. Portanto,

$$E\left[\nabla \mathrm{e}(\hat{y}_i, y_i)\right] = \frac{1}{N}\sum_{i=1}^{N} \nabla \mathrm{e}(\hat{y}_i, y_i) = \nabla \mathcal{L}(\mathbf{w})$$

Agora, como nessa versão estocástica atualizamos os parâmetros a cada exemplo avaliado, a vantagem de aceleramento no processo de treinamento é evidente. E, seguindo essa mesma linha, há, também o método do **Gradiente Descendente *Mini batch*** — que utilizamos na implementação de Rede Neural.

Essa versão do método de otimização consiste na formação de pequenos (*mini*) lotes (*batches*) de dados para a atualização dos parâmetros do modelo. Então, definimos o tamanho de cada lote (k, por exemplo). Logo, a cada época, atualizamos o modelo $\frac{N}{k}$ vezes. Perceba que também há uma natureza estocástica nesse processo, contudo, considerando mais exemplos (no lugar de um único), conseguimos reduzir algumas flutuações aleatórias que estariam presentes no método puramente estocástico. Obtendo, dessa maneira, uma descida mais "suave" do gradiente.

6.7.3.2 MNIST

Para testarmos os conceitos sobre Redes Neurais que foram estudados, recorremos ao conjunto de dados MNIST (*Modified National Institute of Standards and Technology database*). Tal conjunto compila 70000 imagens de dígitos escritos à mão. Há 60000 imagens para o conjunto de treinamento e 10000 imagens para teste (todas com seus respectivos rótulos). É válido mencionar que, esse conjunto de dados é, comumente usado como *Hello World* de aplicações em Machine Learning.

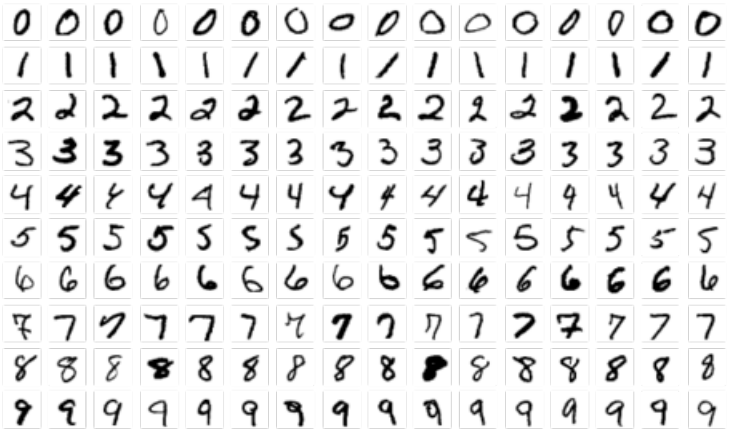

Figura 14: Exemplos de dígitos do MNIST

Seguindo o padrão das demais implementações, utilizamos o Google Collab para fazer os códigos em um notebook e, no arquivo `.ipynb`, há comentários acerca da implementação e dos resultados obtidos.

É possível acessar esse notebook através do seguinte link: NOTEBOOK MNIST

7 Histórico das Atividades

Para relatar como as atividades apresentadas na proposta geral do projeto da disciplina MAC0215 foram realizadas, segue o seguinte histórico:

- Estudo de Regressão Linear
  de 30/08 até 15/09 - totalizando 11 horas
- Estudo de Regressão Logística
  de 20/09 até 15/10 - totalizando 19 horas
- Estudo da função Softmax
  de 15/10 até 07/11 - totalizando 16 horas
- Estudo de Redes Neurais
  de 05/11 até 13/11 - totalizando 12 horas
- Estudo de backpropagation
  de 13/11 até 24/11 - totalizando 10 horas
- Implementação do MNIST from Scratch
  de 15/11 até 27/11 - totalizando 18 horas
- Implementação de Regressão Linear from Scratch
  de 28/11 até 02/11 - totalizando 8 horas
- Implementação de Regressão Logística from Scratch
  de 28/11 até 03/12 - totalizando 6 horas

Referências Aprendizado Baseado em Dados

[1] Yaser S. Abu Mostafa, Magdon Ismail, and Hsuan-Tien Lin. *Learning From Data*. AML, 2012.

[2] Nina S. T. Hirata. Notas de aula de Introdução ao Aprendizado de Máquina, Maio 2022. Universidade de São Paulo.

[3] Martha Salerno Monteiro. Cálculo II - Aula 14 - Parte 3 - Vetor gradiente e derivada direcional de uma função de duas variáveis. Online; postado em 05 de Novembro de 2013; disponível em `https://www.youtube.com/watch?v=rmTs85P0vDM&t`. Universidade de São Paulo.

Referências Regressão Linear

[1] Nina S. T. Hirata. Notas de aula de Introdução ao Aprendizado de Máquina, Maio 2022. Universidade de São Paulo.

[2] Zixuan Zhang. Understand Data Normalization in Machine Learning. Online; postado em 27 de Março de 2019; disponível em `https://towardsdatascience.com/understand-data-normalization-in-machine-learning-8ff3062101f0`.

Referências Regressão Logística

[1] Stéphanie M. van den Berg. Generalised linear models: logistic regression. In *Analysing Data Using Linear Models*. bookdown, 2022.

[2] Jason Brownlee. A Gentle Introduction to Logistic Regression With Maximum Likelihood Estimation. Online; postado em 28 de Outubro de 2019; dsponível em `https://machinelearningmastery.com/logistic-regression-with-maximum-likelihood-estimation/`.

[3] Wikipedia. Likelihood function — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Likelihood%20function&oldid=1123377241`, 2022. [Online; accessed 13-December-2022].

[4] André Zibetti. Distribuição conjunta. Online; disponível em `https://www.inf.ufsc.br/~andre.zibetti/probabilidade/dist_conjunta_vad.html`. Universidade Federal de Santa Catarina.

[5] Puja P. Pathak. Logistic Regression and Maximum Likelihood Estimation Function. Online; postado em 09 de Abril de 2021 ; disponível em https://medium.com/codex/logistic-regression-and-maximum-likelihood-estimation-function-5d8d998245f9.

[6] Arun Addagatla. Maximum Likelihood Estimation in Logistic Regression. Online; postado em 26 de Abril de 2021; disponível em https://arunaddagatla.medium.com/maximum-likelihood-estimation-in-logistic-regression-f86ff1627b67.

[7] Carlos Guestrin. Slides do curso de Machine Learning, Abril 2013. Online; disponível em https://courses.cs.washington.edu/courses/cse446/13sp/slides/logistic-regression-gradient.pdf. Universidade de Washington.

Referências Regressão Multinomial

[1] Looi Kian Seong. Softmax Regression. Online; postado em 16 de Maio de 2020; disponível em https://medium.datadriveninvestor.com/softmax-regression-bda793e2bfc8.

[2] Thomas Phan. A Brief Tutorial on the Relationship Between Maximum Likelihood Estimation and Cross-Entropy Loss in machine learning. 2021. Online; disponível em http://www.octoberraindrops.com/publications/Phan_2021_MLE_vs_Cross_Entropy_Loss.pdf.

[3] Arash Ashrafnejad. Softmax and Cross Entropy Gradients for Backpropagation. Online; postado em 17 de Março de 2020; disponível em https://www.youtube.com/watch?v=5-rVLSc2XdE.

Referências Redes Neurais

[1] Nina S. T. Hirata. Notas de aula de Introdução ao Aprendizado de Máquina, Maio 2022. Universidade de São Paulo.

[2] Michael Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[3] Shivaram Kalyanakrishnan. The Perceptron Learning Algorithm and its Convergence, Janeiro 2017. Indian Institute of Technology Bombay.

[4] Wikipedia. NAND logic — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=NAND%20logic&oldid=1104410550, 2022. [Online; accessed 04-December-2022].

[5] Pragati Baheti. The Essential Guide to Neural Network Architectures. Online; postado em 21 de Outubro de 2022; disponível em https://www.v7labs.com/blog/neural-network-architectures-guide.

[6] Yaser S. Abu Mostafa, Magdon Ismail, and Hsuan-Tien Lin. *Learning From Data*. AML, 2012.