Maqueise Pinheiro

Perceptron Multicamadas: uma ferramenta de Aprendizado Supervisionado

Niterói - RJ, Brasil

20 de setembro de 2021

Universidade Federal Fluminense

Maqueise Pinheiro

Perceptron Multicamadas: uma ferramenta de Aprendizado Supervisionado

Trabalho de Conclusão de Curso

Monografia apresentada para obtenção do grau de Bacharel em Estatística pela Universidade Federal Fluminense.

Orientador: Prof. Dr. Douglas Rodrigues

Niterói - RJ, Brasil

20 de setembro de 2021

Resumo

O modelo perceptron proposto por Rosenblatt (1958), é um modelo de aprendizado de máquinas utilizado tanto em classificação como em regressão. Inserido na técnica de redes neurais artificiais, ele possui uma estrutura composta de neurônios (unidades de processamento dos dados) e camadas (etapas do modelo) que permite o refinamento do resultado de saída. Este trabalho busca entender o modelo perceptron e sua forma mais geral, o perceptron multicamadas. Para isso, descreve cada uma de suas etapas e os elementos que o compõe, desde sua origem inspirada em um dos primeiros modelos de redes neurais, o modelo de McCulloch e Pitts (1943), até a utilização do método do gradiente descendente para melhoria dos parâmetros do modelo, ilustrando também duas funções comuns no papel de função de ativação. A proposta do trabalho foi elaborar um algoritmo perceptron multicamadas no *software R* (R Core Team, 2014) a fim de testar empiricamente o ganho em acrescentar neurônios e/ou camadas à camada oculta em termos de precisão. Para isso, foi utilizada a base de dados *mushrooms* do artigo de Knopf (1981), onde vimos que a adição de camadas assim como a adição de neurônios não implica necessariamente na melhora do modelo em relação a precisão, além de se tornar cada vez mais custoso computacionalmente.

Palavras-chave: Perceptron. Perceptron Multicamadas. Gradiente Descendente. Função de Ativação.

Agradecimentos

Primeiramente agradeço a Deus por estar a meu lado e cuidar de mim em todos os aspectos por mais mundanos que sejam. Sem Ele eu não poderia fazer metade do que já fiz e do que ainda alcançarei com Ele.

Agradeço a minha mãe por todo o suporte que me deu durante esse período e pela amizade perene. Também agradeço meu pai por acreditar que sou capaz de coisas grandes. Agradeço minha irmã por me lembrar que precisamos de música para trabalhar. Agradeço ainda meus outros dois irmãos que me mostram novos mundos e estão sempre presentes. Agradeço também minha tia que me ensinou que estudar e trabalhar não é tudo na vida.

Agradeço especialmente ao Douglas, meu orientador que levo pra vida com carinho, que me ofereceu uma oportunidade em 2017 e depois outra e trabalhamos juntos desde então. Estimo todas as orientações e tempo dedicado.

Agradeço também a professora Karina que fez parte do meu desenvolvimento acadêmico mostrando direções. Assim como o professor Jony que desde o primeiro período tem caminhado comigo. Juntamente com todos os professores que compartilharam seu aprendizado me fazendo hoje uma profissional.

Agradeço a minha amiga Julliany, que me suporta e me da forças. Também aos meus amigos da UFF, Larissa, Isabelle, Gabriel e Larissa que foram essenciais. Com carinho, agradeço a Lucas, meu namorado, que me ajudou e continua ajudando imensamente em tudo que faço. Não tenho palavras que traduzam minha admiração, carinho e respeito. Obrigada por me lembrar que sou capaz.

Sumário

Lista de Figuras

1 Introdução

Em 1943, o neurocientista McCulloch juntamente com o matemático Pitts, propuseram um modelo de rede neural artificial inspirado em um neurônio biológico. Em 1957, o cientista da computação Frank Rosenblatt desenvolveu o modelo de classificação binária perceptron, composto pela estrutura de rede de McCulloch-Pitts dada por $rede = \sum_l x_{i,l}\omega_l + \theta$ de apenas um neurônio, a função degraus de Heaviside como função de ativação e a regra de aprendizado gradiente descendente para correção dos parâmetros de peso e viés (ω_l e θ respectivamente, com l=1,...,p).

O perceptron lida com problemas linearmente separáveis, já que a rede é uma combinação linear dos valores de entrada que gera um hiperplano. Para corrigir essa limitação surgiu o perceptron multicamadas, que possibilita gerar mais de um hiperplano, generalizando o perceptron para problemas de classificação com duas ou mais categorias não necessariamente separáveis linearmente. Ele consiste em usar uma ou mais camadas entre a de entrada e a de saída, denominadas camadas ocultas, que possuem um ou mais neurônios. Além disso, a quantidade de neurônios da camada de saída é relacionado ao número de categorias de respostas possíveis.

Como função de ativação, é comumente usada a função sigmoidal e os parâmetros são atualizados pelo método de retropropagação e gradiente descendente, até que se obtenha um erro quadrático menor que um limite definido. Cada uma dessas etapas é descrita ao longo do texto.

Esse trabalho introduz o conceito de redes neurais artificiais (seção 2.1) com o modelo proposto por McCulloch e Pitts (1943) como preparação para a apresentação do modelo perceptron de Rosenblatt (1958). Para isso, apresentaremos as funções de ativação (seção 2.2), em especial a função degrau de Heaviside e a função sigmoidal. Em seguida, descreveremos o método do gradiente descendente (seção 2.3), em quais casos ele pode ser usado e como utilizá-lo para ajustar os parâmetros do modelo seguindo a retropropagação. Finalmente, apresentaremos uma generalização do modelo, o perceptron multicamadas, que

pode lidar com problemas mais complexos, desde classificação com mais de duas classes até de aprendizado profundo (*deep learning*). Um dos objetivos deste trabalho é avaliar o comportamento do preditor ao variarmos a quantidade de camadas e neurônios no modelo. Para isso, com um intuito de realizar um estudo empírico, construímos um algoritmo no software R (R Core Team, 2014) para aplicar o perceptron multicamadas em um banco de dados e inferir sobre os resultados.

1.1 Motivação

O aprendizado de máquinas é um tema amplamente discutido atualmente. A facilidade em resolver problemas complexos e a gama de modelos disponíveis para os mais variados problemas atrai o interesse de acadêmicos e pesquisadores.

Um desses modelos que merece destaque é o perceptron multicamadas apresentado por Rosenblatt (1958) baseado no trabalho de McCulloch e Pitts (1943). Esse método se sobressai pela eficiência e fácil entendimento, como é passado mais a frente. A sua relevância no meio de aprendizado de máquinas foi o motivo de ter sido escolhido para esse trabalho já que foi um dos modelos pioneiros nesta temática e ainda se mostra satisfatório.

Esse trabalho é motivado pela seção 5 do primeiro capítulo do livro de Mello e Ponti (2018) e busca entender como a adição de neurônios e camadas afeta um modelo de perceptron multicamadas quanto a sua precisão, tempo de processamento e eficiência comparado a outros modelos perceptron multicamadas.

1.2 Objetivos

Compreender o modelo perceptron e perceptron multicamadas em todas as suas etapas tendo como referência inicial a seção 1.5 do livro *Machine learning: a practical approach on the statistical learning theory* dos autores Mello e Ponti (2018).

Comparar modelos perceptron multicamadas de variadas quantidades de camadas ocultas e avaliá-los quanto a sua adequação aos dados, custo computacional e ganho de precisão.

Comparar modelos perceptron multicamadas de variadas quantidades de neurônios na camada oculta e avaliá-los quanto a sua adequação aos dados, custo computacional e

ganho de precisão.

Inferir sobre a quantidade de neurônios e camadas ideais para o ajuste de um modelo à determinadas bases de dados.

1.3 Organização

No capítulo 2, materiais e métodos, podemos encontrar o desenvolvimento do trabalho.

Na seção 2.1, é introduzido o conceito de redes neurais artificiais como uma técnica de aprendizado de máquinas. A partir dela, apresentamos o modelo de McCulloch e Pitts (1943).

Na seção 2.2, é exibido o conceito de funções de ativação e apresentado dois exemplos da mesma.

Na seção 2.3, é mostrado como funciona o método do gradiente descendente e onde é usado.

Na seção 2.4, é apresentado o modelo perceptron de Rosenblatt (1958). Nela vemos como o modelo é esquematizado e aplicado, explorando cada etapa.

Na seção 2.5, o modelo perceptron é generalizados para múltiplas camadas. Nessa seção também vemos a finalidade de cada tipo de camada, como os parâmetros são ajustados e também como o algoritmo se comporta.

No capítulo 3, temos os resultados obtidos de testes feitos em uma base de dados comparando modelos com variadas quantidades de neurônios e camadas.

Por fim, no capítulo 4 vemos as conclusões obtidas a partir desses testes sobre o ajuste ideal de um modelo.

2 Materiais e Métodos

2.1 Redes Neurais Artificiais

Uma rede neural artificial é uma técnica computacional inspirada na estrutura de atividade cerebral humana. A transmissão dos impulsos nervosos no cérebro se dá pela interação entre neurônios por meio da sinapse. O impulso é recebido pelos dendritos, passa pelo corpo celular e é liberado pelo axônio. Da mesma forma, uma rede neural artificial é uma estrutura de camadas, onde a camada de entrada recebe um sinal que passa por unidades de processamento na camada oculta. Essas unidades são chamadas neurônios artificiais. E em seguida, a camada de saída transmite um sinal de resposta.

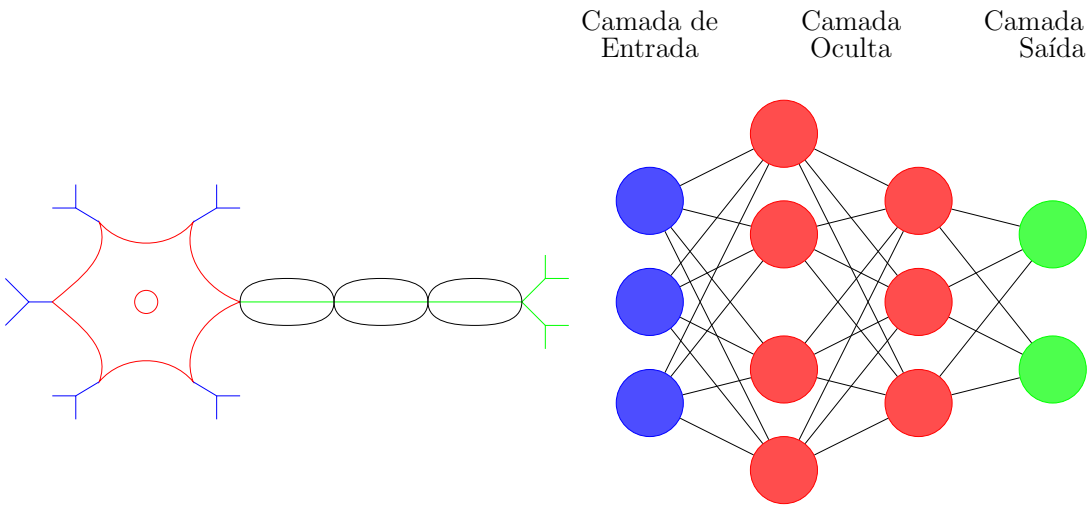

Figura 1: Neurônio Biológico x Rede Neural Artificial

Podemos ver cada saída de um neurônio como a entrada de outro na camada seguinte. Apesar de se chamar "camada oculta"podemos ter mais de uma camada nessa parte da rede. Quando temos uma rede neural de muitas camadas, temos então uma rede neural profunda.

2.1.1 Modelo de McCulloch e Pitts

Em 1943, o neurocientista Warren S. McCulloch juntamente com o matemático Walter Pitts propuseram em seu artigo, McCulloch e Pitts (1943), um modelo de rede neural artificial que entrega como saída uma resposta booleana (ou seja, 0's e 1's).

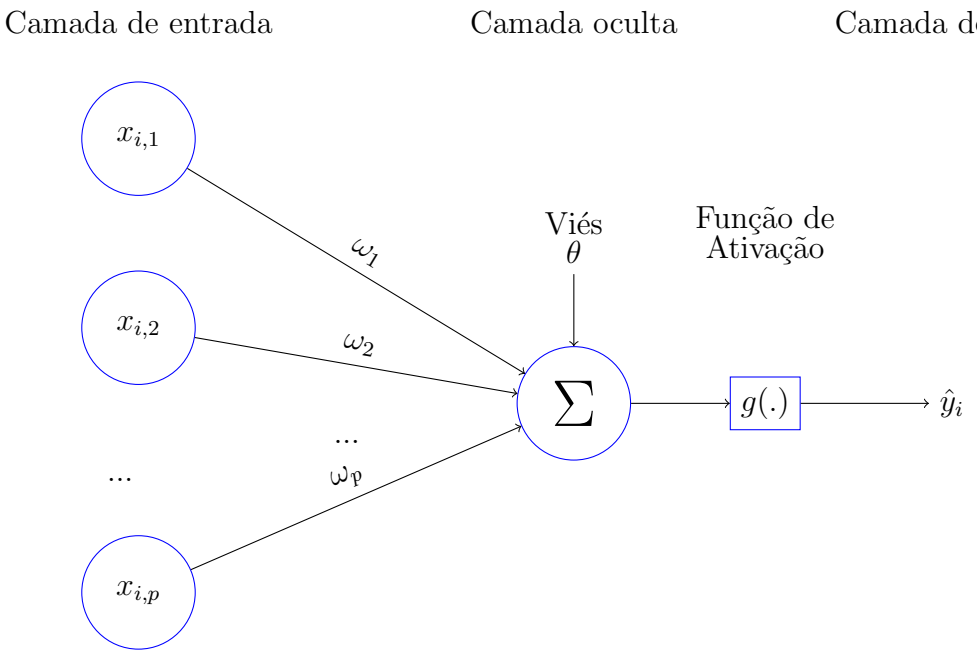

Figura 2: Modelo de McCulloch e Pitts

Nesse modelo, a soma ponderada dos valores das p variáveis de entrada mais um viés (θ) adicionado, produz um valor que é classificado como $\hat{y}_i = 1$ se passa de um limiar (ϵ) e $\hat{y}_i = 0$ caso contrário.

No modelo de McCulloch e Pitts, essa função que transforma a saída em um valor booleano ($g(.)$) é a função de Heaviside, e inserida no modelo é conhecida como **Função de Ativação**.

2.2 Função de Ativação

As funções de ativação aplicam uma transformação não linear nos dados, permitindo soluções de problemas mais complexos que uma regressão linear não seria capaz de resolver. A seguir estão descritas as funções de ativação utilizadas nesse trabalho.

2.2.1 Função Degrau de Heaviside

A função degrau elaborada pelo matemático, engenheiro elétrico e físico Oliver Heaviside atribui valor 1 para o argumento com valor acima de um limiar definido pelo pesquisador (ϵ) e 0 caso contrário.

$$g(x) = \begin{cases} 1, & \text{se x} > \epsilon \\ 0, & \text{c.c.} \end{cases}$$

Produzindo assim um resultado binário.

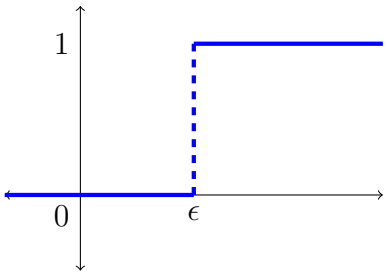

Figura 3: Função Degrau de Heaviside

O parâmetro ϵ é definido pelo pesquisador de forma que se ajuste bem a seus dados, utilizando por exemplo o método de validação cruzada ou da curva ROC (veja mais sobre curva ROC no apêndice 2).

2.2.2 Função Sigmoidal

Também conhecida como função logística, a função sigmoidal é não linear e produz saídas contínuas com valores entre 0 e 1.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

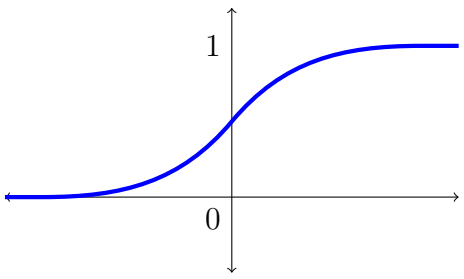

Figura 4: Função Sigmoidal

A função sigmoidal se destaca em problemas de classificação já que ao produzir resultados positivos e limitados ente 0 e 1 facilita a partição dos resultados em duas classes

por meio de um critério definido pelo pesquisador.

Além disso, ela também é diferenciável em todo seu domínio, o que é muito útil no perceptron devido a utilização de derivadas no método gradiente descendente.

2.3 Gradiente descendente

É um método iterativo utilizado para encontrar o valor dos parâmetros que minimizam uma função de interesse, geralmente uma função de erro, dado por

$$\beta\left(t+1\right)=\beta\left(t\right)-\eta\cdot\frac{\partial f}{\partial\beta}$$

Ele consiste em usar o parâmetro atual $\beta\left(t\right)$ menos a derivada parcial da função de interesse f em relação ao parâmetro, ponderada por uma taxa de aprendizado η para definir um novo valor ao β. O parâmetro é atualizado dessa forma até que f atinja a convergência. Para isso é necessário um valor arbitrário inicial para $\beta\left(0\right)$ e definir uma taxa de aprendizado apropriada. Também é recomendado que a função f seja convexa para evitar que o método encontre um mínimo local e não global. Sendo uma função convexa, o gradiente descendente garante encontrar o mínimo.

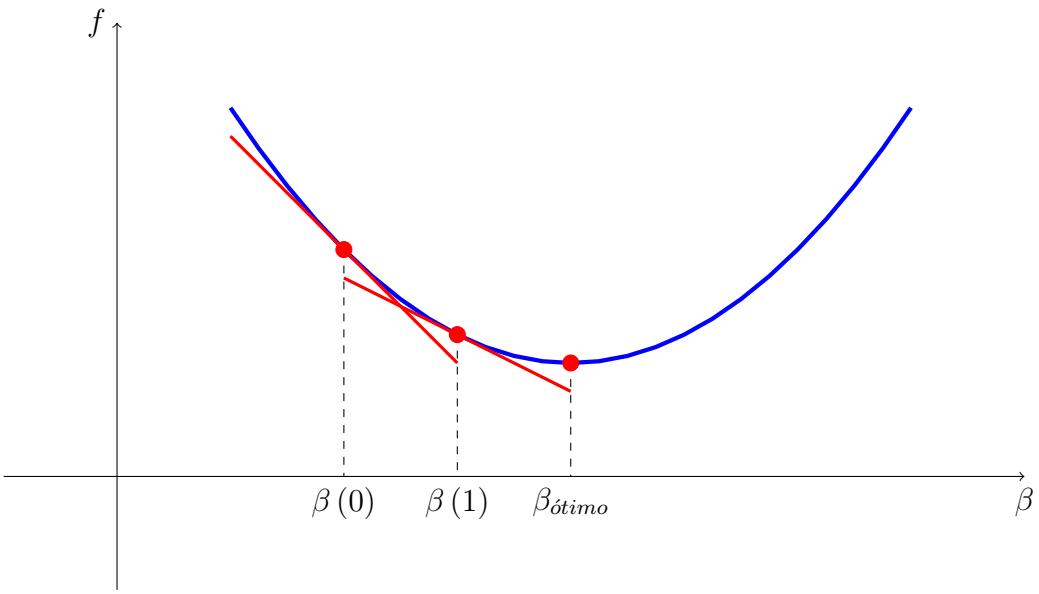

Figura 5: Gradiente Descendente

A ideia é iniciar com um valor qualquer, vamos supor $\beta\left(0\right)$. Se a derivada é negativa, como mostra na figura 5, isso quer dizer que o valor do parâmetro está a esquerda do valor mínimo, então precisamos que ele seja maior. Dessa forma, quando atualizamos e obtemos $\beta\left(1\right)$, ele está um pouco mais próximo do valor mínimo $\beta_{ótimo}$.

Quando a derivada é positiva, significa que o valor atual do parâmetro está a direita do valor mínimo, logo, precisamos diminui-lo.

A taxa de aprendizado vai definir o tamanho do passo que vamos dar em direção ao mínimo. Como ilustra a figura 6, essa taxa não pode ser um valor muito grande, porque o algoritmo vai divergir, nem um valor muito pequeno, porque então custa muito computacionalmente para chegar num resultado aceitável.

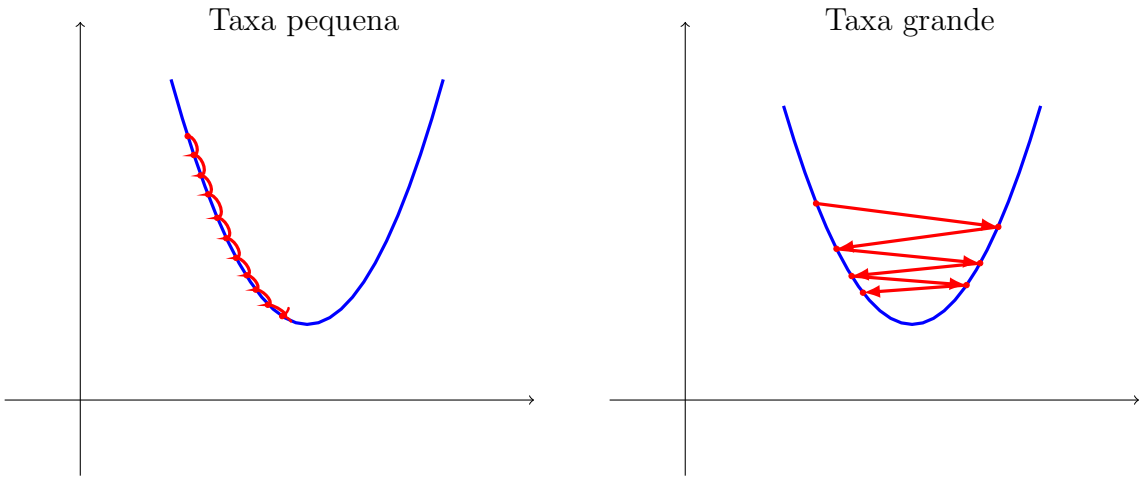

Figura 6: Taxa de Aprendizado

2.4 Perceptron

O algoritmo **perceptron**, proposto por Rosenblatt (1958) é uma rede neural de um único neurônio. Ele é usado em problemas linearmente separáveis já que a rede fornece um hiperplano como resposta.

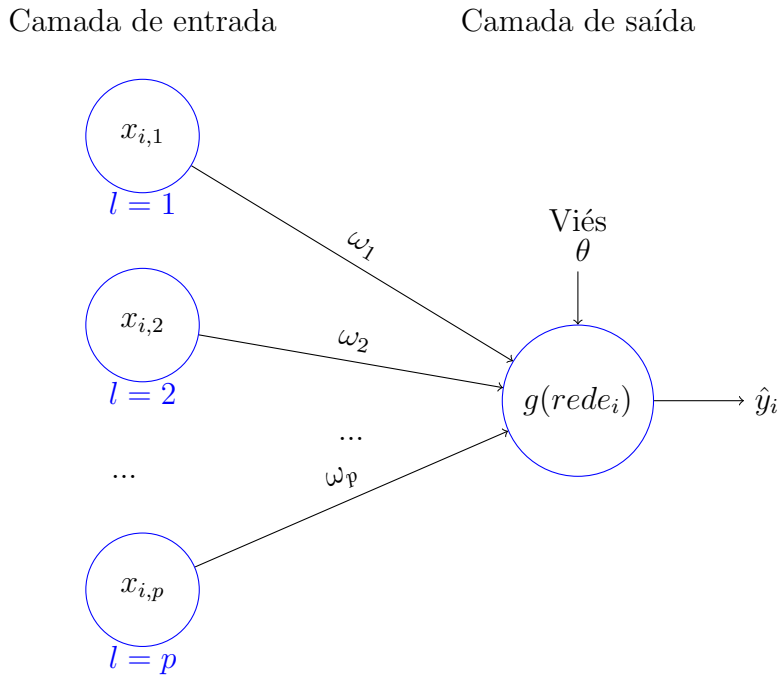

Figura 7: Perceptron

Como é mostrado na figura 7, o modelo recebe um conjunto de dados rotulados na camada de entrada de n observações e p variáveis. É calculado então a rede para cada observação da forma

$$rede_i = \left(\sum_{l=1}^{p} x_{i,l} \cdot \omega_l \right) + \theta, \ com \ i \in \{1, ..., n\}.$$

Onde ω_l, com $l \in \{1, ..., p\}$, são pesos que definem a inclinação do hiperplano e θ é o intercepto.

O resultado da rede serve de argumento para a função de ativação g, que é a função de heaviside, e ela classifica a observação como 0 ou 1.

$$g(rede_i) = \begin{cases} 1, & \text{se } rede_i > \epsilon \\ 0, & \text{c.c.} \end{cases}$$

onde o parâmetro ϵ é definido pelo pesquisador, por exemplo com uma amostra de teste.

Note que a rede é uma função linear. Como estamos lidando com classificação de problemas linearmente separáveis, uma reta de regressão não é a melhor resposta, já que ela vai retornar um número real para cada entrada e queremos uma classificação booleana que nos diga se aquela observação pertence a determinada categoria ou não. Então entra a **função de ativação** aplicando uma transformação não linear nas saídas da rede, nesse caso, classificando todos os valores onde a rede é maior que limite ϵ como 1 e caso contrário

como 0.

Após a obtenção dos n resultados, usamos a soma dos erros quadrados (2.2) para avaliar se o modelo foi bem ajustado. Para isso, definimos um nível de tolerância δ que, estando o valor do E^2, definido na equação 2.2, abaixo de δ, consideramos o modelo bem ajustado.

$$\begin{aligned} E^2 &= \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2 \\ &= \sum_{i=1}^{n} \left(y_i - g\left(rede_i\right)\right)^2 \\ &= \sum_{i=1}^{n} \left(y_i - g\left(\left[\sum_{l=1}^{p} x_{i,l} \cdot \omega_l\right] + \theta\right)\right)^2 \end{aligned} \tag{2.2}$$

Caso o valor de E^2 seja maior que δ, o modelo atualiza seus parâmetros (ω's e θ) usando o método do **Gradiente Descendente** (GD).

Queremos então achar os parâmetros que minimizam E^2. Sendo assim, os parâmetros atualizados pelo GD tem a forma:

$$\omega_s\left(t+1\right) = \omega_s\left(t\right) - \eta \cdot \frac{\partial E^2\left(t\right)}{\partial \omega_s}$$

para $s = 1, ..., p$

$$\theta\left(t+1\right) = \theta\left(t\right) - \eta \cdot \frac{\partial E^2\left(t\right)}{\partial \theta}$$

Mas note que, como a função g possui descontinuidade de salto, sua derivada não existe no ponto ϵ e tem valor 0 fora desse ponto. Por esse motivo, Rosenblatt considerou E^{2*} ao usar o gradiente descendente, onde

$$\begin{aligned} E^{2*} &= \sum_{i=1}^{n} \left(y_i - rede_i\right)^2 \\ &= \sum_{i=1}^{n} \left(y_i - \left(\left[\sum_{l=1}^{p} x_{i,l} \cdot \omega_l\right] + \theta\right)\right)^2 \end{aligned} \tag{2.3}$$

O método funciona bem nesse caso porque a soma dos erros quadrados em E^{2*} se trata de uma função convexa em problemas linearmente separáveis, como ilustrado na figura 8. Pense que temos uma única variável explicativa x e portanto um único peso ω. Suponha também que nesse modelo $\theta = 0$. Assim, quando ω é muito grande ou pequeno, a rede ou hiperplano acaba não passando muito perto dos dados, gerando um E^{2*} alto. Já se o ω é bem ajustado, o E^{2*} se minimiza. Dessa forma, não corremos o risco de encontrar um

mínimo não global. O mesmo vale para o θ e os demais ω's que possam ter no modelo.

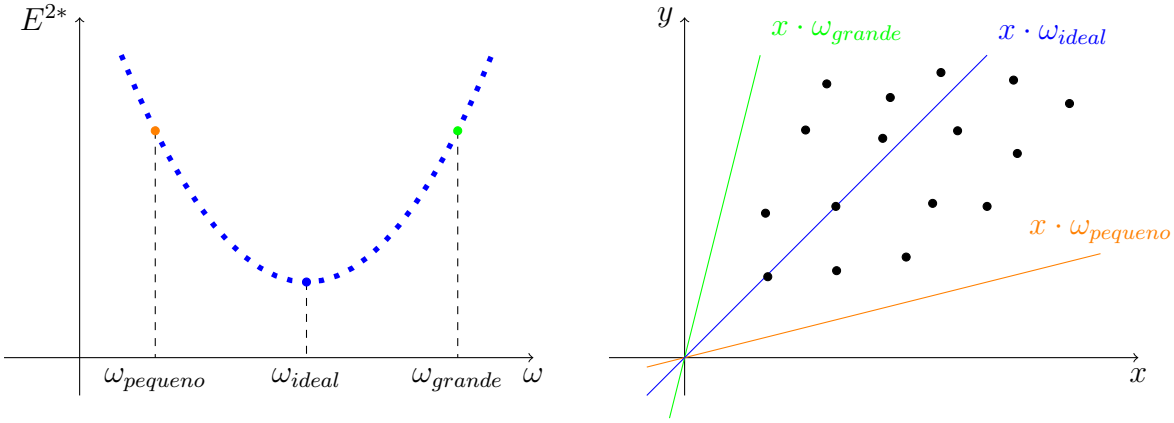

Figura 8: Comportamento convexo do E^{2*} em relação ao parâmetro ω

Dessa forma, podemos calcular a atualização dos pesos como sendo

$$\begin{aligned} \omega_s(t+1) &= \omega_s(t) - \eta \cdot \frac{\partial E^{2*}(t)}{\partial \omega_s} \\ &= \omega_s(t) - \eta \cdot \frac{\partial \sum_{i=1}^{n} (y_i - rede_i(t))^2}{\partial \omega_s} \end{aligned} \tag{2.4}$$

Veja que

$$\begin{aligned} \frac{\partial \sum_{i=1}^{n} (y_i - rede_i(t))^2}{\partial \omega_s} &= 2 \cdot \sum_{i=1}^{n} (y_i - rede_i(t)) \cdot \frac{\partial (y_i - rede_i(t))}{\partial \omega_s} \\ &= 2 \cdot \sum_{i=1}^{n} (y_i - rede_i(t)) \cdot \left[\frac{\partial y_i}{\partial \omega_s} - \frac{\partial rede_i(t)}{\partial \omega_s} \right] \\ &= 2 \cdot \sum_{i=1}^{n} (y_i - rede_i(t)) \cdot \left[0 - \frac{\partial rede_i(t)}{\partial \omega_s} \right] \\ &= 2 \cdot \sum_{i=1}^{n} (y_i - rede_i(t)) \cdot \left(-\frac{\partial rede_i(t)}{\partial \omega_s} \right) \end{aligned} \tag{2.5}$$

e também,

$$\begin{aligned} \frac{\partial rede_i(t)}{\partial \omega_s} &= \frac{\partial \left(\left[\sum_{l=1}^{p} x_{i,l} \cdot \omega_l \right] + \theta \right)}{\partial \omega_s} \\ &= \frac{\partial x_{i,s} \cdot \omega_s}{\partial \omega_s} \\ &= x_{i,s} \end{aligned} \tag{2.6}$$

Substituindo 2.6 em 2.5 e 2.5 em 2.4, temos

$$\begin{aligned}\omega_s(t+1) &= \omega_s(t) - \eta \cdot \left[2 \cdot \sum_{i=1}^{n} (y_i - rede_i(t)) \cdot (-x_{i,s})\right] \\ &= \omega_s(t) + \eta \cdot 2 \cdot \sum_{i=1}^{n} (y_i - rede_i(t)) \cdot x_{i,s} \end{aligned} \tag{2.7}$$

Da mesma forma, a atualização do viés fica

$$\begin{aligned}\theta(t+1) &= \theta(t) - \eta \cdot \frac{\partial E^{2*}(t)}{\partial \theta} \\ &= \theta(t) - \eta \cdot \frac{\partial \sum_{i=1}^{n} (y_i - rede_i(t))^2}{\partial \theta} \end{aligned} \tag{2.8}$$

Note que

$$\begin{aligned}\frac{\partial \sum_{i=1}^{n} (y_i - rede_i(t))^2}{\partial \theta} &= 2 \cdot \sum_{i=1}^{n} (y_i - rede_i(t)) \cdot \frac{\partial (y_i - rede_i(t))}{\partial \theta} \\ &= 2 \cdot \sum_{i=1}^{n} (y_i - rede_i(t)) \cdot \left[\frac{\partial y_i}{\partial \theta} - \frac{\partial rede_i(t)}{\partial \theta}\right] \\ &= 2 \cdot \sum_{i=1}^{n} (y_i - rede_i(t)) \cdot \left[0 - \frac{\partial rede_i(t)}{\partial \theta}\right] \\ &= 2 \cdot \sum_{i=1}^{n} (y_i - rede_i(t)) \cdot \left(-\frac{\partial rede_i(t)}{\partial \theta}\right) \end{aligned} \tag{2.9}$$

e também,

$$\begin{aligned}\frac{\partial rede_i(t)}{\partial \theta} &= \frac{\partial ([\sum_{l=1}^{p} x_{i,l} \cdot \omega_l] + \theta)}{\partial \theta} \\ &= \frac{\partial \sum_{l=1}^{p} x_{i,l} \cdot \omega_l}{\partial \theta} + \frac{\partial \theta}{\partial \theta} \\ &= 0 + 1 \\ &= 1 \end{aligned} \tag{2.10}$$

Substituindo 2.10 em 2.9 e 2.9 em 2.8, temos

$$\begin{aligned}\theta(t+1) &= \theta(t) - \eta \cdot \left[2 \cdot \sum_{i=1}^{n} (y_i - rede_i(t)) \cdot (-1)\right] \\ &= \theta(t) + \eta \cdot 2 \cdot \sum_{i=1}^{n} (y_i - rede_i(t)) \end{aligned} \tag{2.11}$$

Após a atualização dos parâmetros, passaremos os dados pelo modelo ilustrado na

figura 7 novamente. Se ao final a soma dos erros quadrados E^2 estiver menor ou igual a δ, então consideramos os parâmetros usados nessa iteração como os gerados pelo perceptron para a classificação de novos dados não rotulados. Caso contrário, continuamos a atualizar os parâmetros a cada iteração até que se atinja um E^2 convenientemente baixo.

É possível que em alguns casos o E^2 estabilize em um valor acima do que se espera. Para evitar o gasto computacional desnecessário, podemos definir um outro critério de parada como um número máximo de iterações a serem consideradas caso o E^2 não alcance o valor desejado.

Uma observação importante é que a soma dos erros quadrados modificada E^{2*} só deve ser usada no método do gradiente descendente. Para a avaliação do modelo, usamos a soma dos erros quadrados não modificada E^2.

A vantagem de usar o perceptron é que ele consegue lidar com problemas não lineares devido à função de ativação. No entanto, ele retorna sempre uma variável resposta de apenas duas categorias.

2.5 Perceptron Multicamadas (MLP)

Por vezes precisamos lidar com problemas que não são linearmente separáveis. Podemos também ter uma variável de interesse que possui mais de duas categorias ou ainda ser contínua.

Por esse motivo, foi desenvolvido o **perceptron multicamadas** que além da camada de entrada e saída, possui uma ou mais camadas ocultas. Veja um exemplo de representação na figura 9.

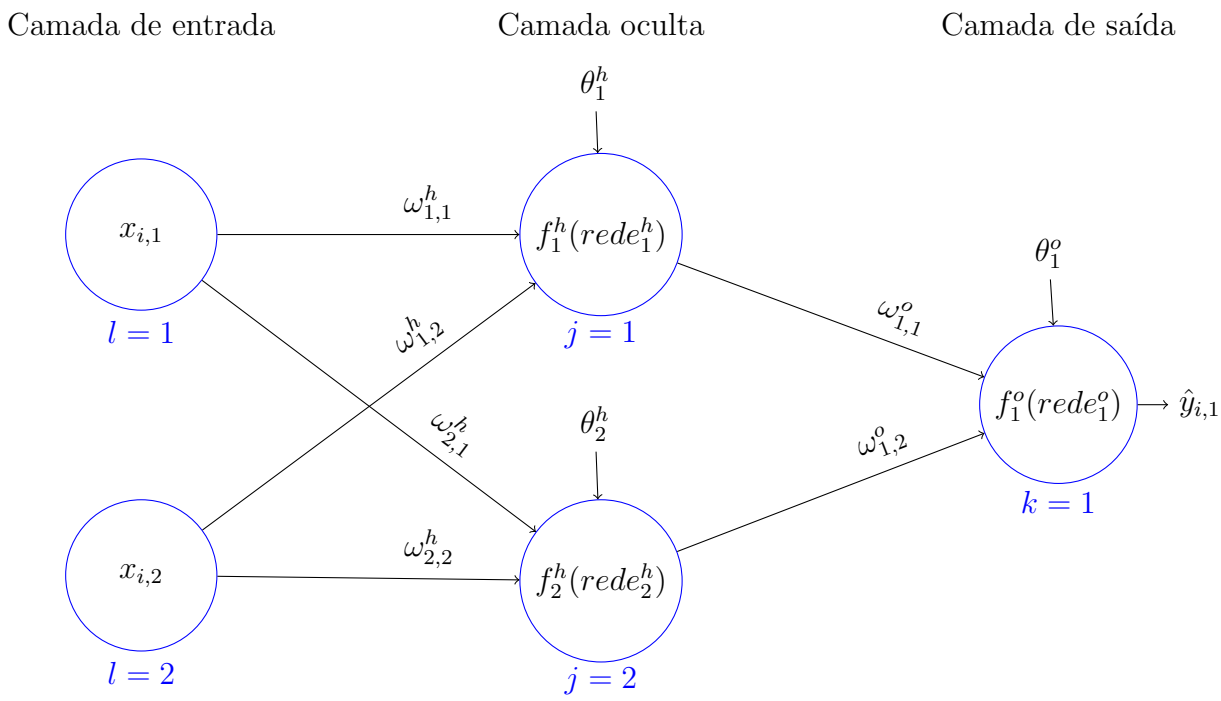

Figura 9: Exemplo de Perceptron Multicamadas

Nesse exemplo da figura 9, note que temos dois tipos de rede.

A $rede_j^h$ é como vimos no perceptron simples. Ela recebe os dados de entrada que são ponderados e somados juntamente com um viés. Ela leva, assim como os parâmetros $\omega_{j,l}^h$ e θ_j^h, com $j = \{1, ..., h\}$, o sobrescrito h indicando que é calculada na camada oculta (*hidden layer*). Note também, que os parâmetros de peso agora possuem a notação $\omega_{j,l}$ indicando que pondera o dado que veio do neurônio l para a rede do neurônio j. Semelhantemente, o viés θ_j está contribuindo para o neurônio j.

$$rede_j^h = \left[\sum_{l=1}^{p} x_{i,l} \cdot \omega_{j,l}^h\right] + \theta_j^h$$

Já a $rede_k^o$ recebe a saída dos neurônios $j's$ anteriores, dessa forma, ela utiliza os resultados das funções de ligação f_j^h, com $j = \{1, ..., h\}$, dos neurônios anteriores como argumento de entrada. O sobrescrito o indica que é calculada na camada de saída (*output layer*). E semelhantemente, os parâmetros de peso possuem a notação $\omega_{k,j}$ indicando que pondera o dado que veio do neurônio j para a rede do neurônio k.

$$rede_k^o = \left[\sum_{j=1}^{h} f_j^h(rede_j^h) \cdot \omega_{k,j}^o\right] + \theta_k^o$$

Dessa forma, o modelo constrói um hiperplano para cada rede criada na camada

oculta, podendo assim lidar com os casos não linearmente separáveis.

Também podemos ter mais de um neurônio da camada de saída, e dessa forma, conseguimos classificar os dados em mais de duas categorias.

Nesse modelo, trabalhamos com funções de ativações mais complexas. Essa função será uma que consiga produzir a saída que desejamos. Assim como no *perceptron* simples que usávamos a função de heaviside para produzir resultados booleanos (0's e 1's), nesse caso a nossa f será a função sigmoidal que consegue produzir resultados quantitativos contínuos entre 0 e 1 (veja a figura 4)

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

2.5.1 As camadas

A **camada de entrada** deve ter a quantidade de neurônios equivalente à quantidade de variáveis explicativas que serão usadas. Esses neurônios serão numerados e identificados pela letra l, onde $l = \{1, ..., p\}$ e p é o número de variáveis explicativas da amostra de treino. Ela não possui parâmetros pois a função desses neurônios é a própria função identidade.

Já na **camada oculta**, a quantidade de neurônios determina a quantidade de hiperplanos que serão construídos no modelo. Eles serão identificados pela letra j, sendo que $j = \{1, ..., h\}$ onde h é a quantidade de hiperplanos que serão construídos. Os parâmetros utilizados na função de ativação f_j^h em cada neurônio serão identificados pelo sobrescrito h, assim como a própria rede. É possível ter mais de uma camada na camada oculta.

A **camada de saída**, num problema de classificação, possui o número de neurônios necessários para produzir a resposta de um jeito semelhante às variáveis indicadoras (*dummies*). Assim, se a variável de interesse é por exemplo faixa etária e possui 4 categorias: "criança", "adolescente", "adulto"e "idoso". Escolhemos uma para ser a categoria base, digamos "idoso". Assim, teremos 3 neurônios que produzirão resultados capazes de responder as perguntas:

neurônio k=1: "O indivíduo é criança?"

neurônio k=2: "O indivíduo é adolescente?"

neurônio k=3: "O indivíduo é adulto?"

E se em todas as 3 perguntas obtivermos resposta "não", então aquele indivíduo será classificado como "idoso". Note que os grupos são exclusivos, isto é, um indivíduo não pode estar presente em mais de uma faixa etária ao mesmo tempo.

Cada neurônio nessa camada é identificado pela letra k, sendo $k = \{1, ..., c\}$ onde c é o número de categorias da variável resposta menos 1. Os parâmetros utilizados na função de ativação f_k^o em cada neurônio serão identificados pelo sobrescrito o, assim como a própria rede.

Note também que a função de ativação é responsável pelo tipo de saída que vamos obter. Usando a função sigmoidal temos saídas em cada neurônio k com valores entre 0 e 1. Assim, podemos dizer , por exemplo, que os neurônios com resultados maiores que 0.5 são classificados como 1 e os menores como 0.

Se nosso problema é de regressão, então só precisamos de um neurônio para produzir resultados contínuos, lembrando sempre de usar a função de ativação mais apropriada.

2.5.2 Atualizando os parâmetros

Da mesma forma que no perceptron simples, atualizamos os parâmetros utilizando o gradiente descendente. A diferença é que agora temos, além dos parâmetros da camada de saída, os parâmetros da camada oculta.

- $\omega_{j,l}^h(t+1) = \omega_{j,l}^h(t) - \eta \cdot \frac{\partial E^2(t)}{\partial \omega_{j,l}^h}$
- $\theta_j^h(t+1) = \theta_j^h(t) - \eta \cdot \frac{\partial E^2(t)}{\partial \theta_j^h}$
- $\omega_{k,j}^o(t+1) = \omega_{k,j}^o(t) - \eta \cdot \frac{\partial E^2(t)}{\partial \omega_{k,j}^o}$
- $\theta_k^o(t+1) = \theta_k^o(t) - \eta \cdot \frac{\partial E^2(t)}{\partial \theta_k^o}$

  onde, $l = 1, ..., p$, $j = 1, ..., h$ e $k = 1, ..., c$.

Vamos utilizar como função de ativação a função sigmoidal. Note que ela é uma função diferenciável, dessa forma, podemos utilizar o E^2, descrito na equação 2.12, sem problemas no GD.

$$
\begin{aligned}
E^2 &= \sum_{i=1}^{n}\sum_{k=1}^{c}\left(y_i-\hat{y}_i\right)^2 \\
&= \sum_{i=1}^{n}\sum_{k=1}^{c}\left(y_i-f_k^o\left(rede_{i,k}^o\right)\right)^2 \\
&= \sum_{i=1}^{n}\sum_{k=1}^{c}\left(y_i-f_k^o\left[\sum_{j=1}^{h}f_j^h\left(rede_{i,j}^h\right)\cdot\omega_{k,j}^o+\theta_k^o\right]\right)^2 \\
&= \sum_{i=1}^{n}\sum_{k=1}^{c}\left(y_i-f_k^o\left[\sum_{j=1}^{h}f_j^h\left(\sum_{l=1}^{p}x_{i,l}\cdot\omega_{j,l}^h+\theta_j^h\right)\cdot\omega_{k,j}^o+\theta_k^o\right]\right)^2
\end{aligned}
\tag{2.12}
$$

Sabemos que a derivada da função sigmoidal (equação 2.1) é dada por

$$
\begin{aligned}
\frac{\partial f(x)}{\partial x} &= -\frac{1}{\left(1+e^{-x}\right)^2}\cdot\left(-e^{-x}\right) \\
&= \frac{e^{-x}+1-1}{\left(1+e^{-x}\right)^2} \\
&= \frac{1+e^{-x}}{\left(1+e^{-x}\right)^2}-\frac{1}{\left(1+e^{-x}\right)^2} \\
&= \frac{1}{\left(1+e^{-x}\right)}-\frac{1}{\left(1+e^{-x}\right)^2} \\
&= \frac{1}{\left(1+e^{-x}\right)}\cdot\left[1-\frac{1}{\left(1+e^{-x}\right)}\right] \\
&= f(x)\cdot\left(1-f(x)\right)
\end{aligned}
\tag{2.13}
$$

Com isso, para o **peso da camada oculta**

$$
\omega_{j,l}^h(t+1) = \omega_{j,l}^h(t)-\eta\cdot\frac{\partial E^2(t)}{\partial\omega_{j,l}^h} \tag{2.14}
$$

onde

$$
\begin{aligned}
\frac{\partial E^2(t)}{\partial\omega_{j,l}^h} = &\sum_{i=1}^{n}\sum_{k=1}^{c}2\cdot\left(y_i-f_k^o[\phi]\right)\cdot\left(-f_k^o[\phi]\cdot\left(1-f_k^o[\phi]\right)\right)\cdot \\
&\omega_{k,j}^o\cdot\left(-f_j^h(\psi)\cdot\left(1-f_j^h(\psi)\right)\right)\cdot x_{i,l}
\end{aligned}
\tag{2.15}
$$

e

- $\phi = f_j^h(\psi)\cdot\omega_{k,j}^o+\theta_k^o$
- $\psi = x_{i,l}\cdot\omega_{j,l}^h+\theta_j^h$

Substituindo 2.15 em 2.14,

$$\begin{aligned}\omega_{j,l}^{h}(t+1) &= \omega_{j,l}^{h}(t) - \eta \cdot \\ &\sum_{i=1}^{n}\sum_{k=1}^{c} 2 \cdot (y_i - f_k^o[\phi]) \cdot (-f_k^o[\phi] \cdot (1 - f_k^o[\phi])) \cdot \\ &\omega_{k,j}^{o} \cdot \left(-f_j^h(\psi) \cdot \left(1 - f_j^h(\psi)\right)\right) \cdot x_{i,l} \end{aligned} \tag{2.16}$$

Da mesma forma, a atualização do **viés da camada oculta** se da por,

$$\theta_j^h(t+1) = \theta_j^h(t) - \eta \cdot \frac{\partial E^2(t)}{\partial \theta_j^h} \tag{2.17}$$

onde

$$\begin{aligned}\frac{\partial E^2(t)}{\partial \theta_j^h} &= \sum_{i=1}^{n}\sum_{k=1}^{c} 2 \cdot (y_i - f_k^o[\phi]) \cdot (-f_k^o[\phi] \cdot (1 - f_k^o[\phi])) \cdot \\ &\omega_{k,j}^{o} \cdot \left(-f_j^h(\psi) \cdot \left(1 - f_j^h(\psi)\right)\right) \cdot 1 \end{aligned} \tag{2.18}$$

e

- $\phi = f_j^h(\psi) \cdot \omega_{k,j}^o + \theta_k^o$
- $\psi = \sum_{l=1}^{p} x_{i,l} \cdot \omega_{j,l}^h + \theta_j^h$

Substituindo 2.18 em 2.17,

$$\begin{aligned}\theta_j^h(t+1) &= \theta_j^h(t) - \eta \cdot \\ &\sum_{i=1}^{n}\sum_{k=1}^{c} 2 \cdot (y_i - f_k^o[\phi]) \cdot (-f_k^o[\phi] \cdot (1 - f_k^o[\phi])) \cdot \\ &\omega_{k,j}^{o} \cdot \left(-f_j^h(\psi) \cdot \left(1 - f_j^h(\psi)\right)\right) \end{aligned} \tag{2.19}$$

Para os **pesos da camada de saída**, temos que

$$\omega_{k,j}^o(t+1) = \omega_{k,j}^o(t) - \eta \cdot \frac{\partial E^2(t)}{\partial \omega_{k,j}^o} \tag{2.20}$$

onde

$$\begin{aligned}\frac{\partial E^2(t)}{\partial \omega_{k,j}^o} &= \sum_{i=1}^{n} 2 \cdot (y_i - f_k^o[\phi]) \cdot (-f_k^o[\phi] \cdot (1 - f_k^o[\phi])) \cdot \\ &f_j^h\left(\sum_{l=1}^{p} x_{i,l} \cdot \omega_{j,l}^h + \theta_j^h\right) \end{aligned} \tag{2.21}$$

e

- $\phi = f_j^h \left(\sum_{l=1}^{p} x_{i,l} \cdot \omega_{j,l}^h + \theta_j^h \right) \cdot \omega_{k,j}^o + \theta_k^o$

Substituindo 2.21 em 2.20,

$$\begin{aligned} \omega_{k,j}^o (t+1) &= \omega_{k,j}^o (t) - \eta \cdot \\ &\sum_{i=1}^{n} 2 \cdot (y_i - f_k^o [\phi]) \cdot (-f_k^o [\phi] \cdot (1 - f_k^o [\phi])) \cdot \\ &f_j^h \left(\sum_{l=1}^{p} x_{i,l} \cdot \omega_{j,l}^h + \theta_j^h \right) \end{aligned} \tag{2.22}$$

E para o **viés da camada de saída**,

$$\theta_k^o (t+1) = \theta_k^o (t) - \eta \cdot \frac{\partial E^2 (t)}{\partial \theta_k^o} \tag{2.23}$$

onde

$$\frac{\partial E^2 (t)}{\partial \theta_k^o} = \sum_{i=1}^{n} 2 \cdot (y_i - f_k^o [\phi]) \cdot (-f_k^o [\phi] \cdot (1 - f_k^o [\phi])) \cdot 1 \tag{2.24}$$

e

- $\phi = \sum_{j=1}^{h} f_j^h \left(\sum_{l=1}^{p} x_{i,l} \cdot \omega_{j,l}^h + \theta_j^h \right) \cdot \omega_{k,j}^o + \theta_k^o$

Substituindo 2.24 em 2.23,

$$\begin{aligned} \theta_k^o (t+1) &= \theta_k^o (t) - \eta \cdot \\ &\sum_{i=1}^{n} 2 \cdot (y_i - f_k^o [\phi]) \cdot (-f_k^o [\phi] \cdot (1 - f_k^o [\phi])) \end{aligned} \tag{2.25}$$

2.5.3 Retropropagação

Agora que já temos as equações obtidas pelo GD, podemos atualizar os parâmetros utilizando retropropagação.

A **retropropagação** (do inglês, *backpropagation*) nada mais é do que ao final de cada iteração, atualizar os parâmetros seguindo a ordem contrária que eles aparecem no algoritmo. Isso é, primeiro os da camada de saída, depois os da última camada da camada oculta e assim sucessivamente até a primeira camada da camada oculta (A camada de entrada não possui parâmetros).

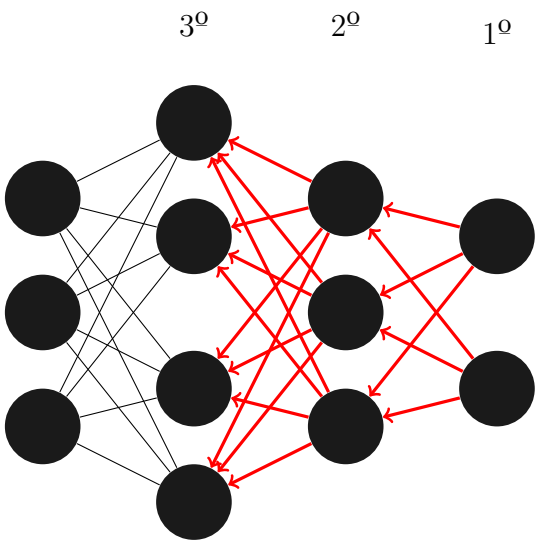

Figura 10: Retropropagação

Fazemos dessa forma para garantir que o método do GD feito na iteração $t+1$ utilize os parâmetros encontrados na iteração t. Se utilizássemos a propagação para frente (*feed-forward*), os parâmetros da camada de saída por exemplo levariam em conta os parâmetros da camada oculta já atualizados.

Por exemplo, na atualização do viés da camada de saída θ_k^o (equação 2.25), a equação leva em conta o viés da camada oculta (θ_j^h). Se utilizássemos a propagação pra frente, os parâmetros seriam atualizados na ordem em que aparecem no modelo, isto é, primeiro os da primeira camada oculta, seguindo até a última camada oculta e por fim os da de saída. Dessa forma, ao calcular o θ_k^o, iríamos utilizar o θ_j^h já atualizado. Utilizando a retropropagação, o θ_j^h só será modificado após o θ_k^o e assim, a iteração atual será toda baseada na iteração anterior.

2.5.4 O Algoritmo

Sendo assim, o algoritmo de perceptron multicamadas se dá por

1. iniciar valores aleatórios para todos os parâmetros (ω's e θ's);
2. passar cada observação pelo modelo;
3. calcular o erro E^2 associado ao modelo;
4. se o erro for maior que um limiar δ pré estabelecido, então atualizamos os parâmetro utilizando GD e retropropagação e voltamos ao passo 2.

3 Análise dos Resultados

Qual é o ganho em acrescentar camadas ocultas em um modelo perceptron multicamadas? Seria melhor acrescentar neurônios numa única camada?

Visando responder essas perguntas, elaboramos um algoritmo no *software R* (R Core Team, 2014) do perceptron multicamadas para testar empiricamente esses conceitos. Para isso, utilizamos uma base de dados obtida do repositório de aprendizado de máquinas da UCI (DUA; GRAFF, 2017) (mais detalhes no apêndice 1).

Após o tratamento e pré-processamento (descritos no apêndice 1), treinamos 18 modelos de classificação utilizando diferentes quantidades de neurônios e camadas em cada um deles. Utilizamos a base ***mushrooms*** do artigo de Knopf (1981). Ela contém 22 características refentes a 8124 cogumelos. O objetivo é saber quais são comestíveis e quais venenosos. Após o tratamento, foram escolhidas 18 variáveis para o treinamento do modelo (veja mais no apêndice 1). Uma amostra de 6093 cogumelos foi separada para treinamento e os 2031 restantes utilizada para teste. Os resultados obtidos podem ser achados na tabela 1.

Tabela 1: Resultados dos modelos da base *mushrooms*

Modelo	Neurônios	Tempo	Iterações	Ponto de corte	AUC	Precisão dentro	Precisão fora	TFP	TVP
1	(2)	00:43:40	10000	0.4978	1	1	1	0	1
2	(3)	00:43:40	10000	0.4985	1	1	1	0	1
3	(5)	00:44:57	10000	0.4963	1	1	1	0	1
4	(2,2)	00:58:34	10000	0.4985	1	1	1	0	1
5	(2,3)	00:57:34	10000	0.499	1	1	1	0	1
6	(2,5)	00:57:58	10000	0.4986	1	1	1	0	1
7	(3,2)	00:58:52	10000	0.4991	1	1	1	0	1
8	(3,3)	00:59:07	10000	0.4991	1	1	1	0	1
9	(3,5)	01:00:31	10000	0.499	1	1	1	0	1
10	(5,2)	00:59:50	10000	0.4985	1	1	1	0	1
11	(5,3)	01:00:44	10000	0.4985	1	1	1	0	1
12	(5,5)	01:00:45	10000	0.4989	1	1	1	0	1
13	(2,2,2,2,2)	01:37:59	10000	0.5085	0.9887	0.9902	0.9902	0	0.9796
14	(3,3,3,3,3)	01:40:56	10000	0.5089	0.9869	0.9902	0.9902	0	0.9796
15	(5,5,5,5,5)	01:45:53	10000	0.4994	1	1	1	0	1
16	(2,2,2,2,2,2,2,2,2,2)	02:50:23	10000	0.5302	0.8219	0.7684	0.482	1	1
17	(3,3,3,3,3,3,3,3,3,3)	02:48:11	10000	0.5302	0.7002	0.3323	0.482	1	1
18	(5,5,5,5,5,5,5,5,5,5)	02:51:50	10000	0.5302	0.6574	0.3901	0.482	1	1

Nessa tabela podemos encontrar a quantidade de neurônios em cada camada de cada modelo. Por exemplo, o modelo 11 tem duas camadas ocultas, a primeira com 5 neurônios e a segunda com 3. Para todos os modelos dessa base, foram utilizados 87 neurônios que é a quantidade de variáveis explicativas da base de dados depois da transformação das variáveis categóricas em indicadoras. Na camada de saída, temos apenas 1 neurônio já que nossa classificação é binária (1 para cogumelos venenosos e 0 para comestíveis). Também podemos ver o tempo que foi necessário para treinar cada modelo e o número de iterações feitas.

Nosso modelo utiliza a função de ativação sigmoidal. Por esse motivo, a saída do modelo são valores entre 0 e 1. Assim, precisamos definir um ponto de corte para fazer a classificação dos dados. Para fazer isso, usamos o método da curva ROC (descrito no apêndice 2). O ponto de corte selecionado em cada modelo pode ser visto na tabela 1. Também foi calculado a área sob a curva ROC (AUC - *area under the curve*) para fazer a comparação dos modelos, a precisão dentro e fora da amostra e as taxas de verdadeiros positivos e falsos positivos. Mais sobre essas medidas também pode ser visto no apêndice 2.

4 Conclusões

Podemos ver que na base *mushrooms* o modelo perceptron multicamadas foi muito bem aceito. Nas figuras 11 e 12 vemos a evolução da precisão dentro da amostra (isto é, calculada na amostra de treino) dos modelos 7 e 13, respectivamente, ao longo das iterações. Pelas figuras podemos ver que com bem menos iterações já teríamos atingido uma estacionariedade na precisão, sendo assim, podemos concluir que muitas iterações não implicam em aumento de precisão. Também é importante notar que no modelo 7 (com duas camadas ocultas com 2 e 3 neurônios respectivamente), a precisão final é maior e alcançada mais rápido enquanto no modelo 13 (com 5 camadas ocultas com 2 neurônios em cada uma delas) é necessário algumas iterações a mais pra atingir a precisão final que é um pouco inferior a do modelo 7.

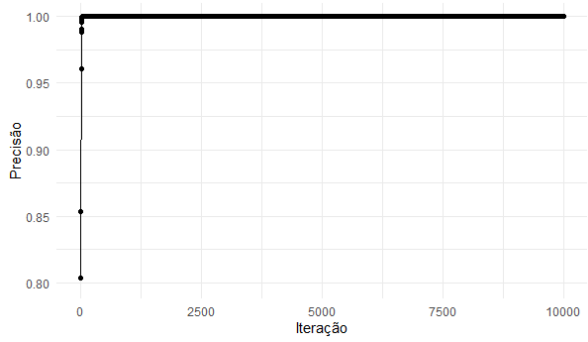

Figura 11: Evolução da precisão no modelo 7

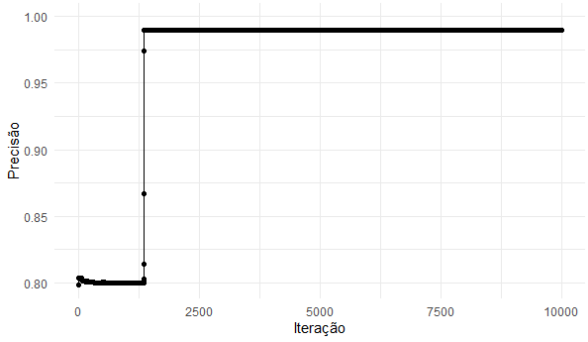

Figura 12: Evolução da precisão no modelo 13

Dos 18 modelos construídos, apenas 3 não tiveram resultados satisfatórios. É curioso como esses 3 modelos foram justamente os que tiveram mais camadas ocultas (10 camadas). Na figura 13 e 14 vemos a evolução da precisão dentro da amostra do modelo 17 (com 10 camadas com 3 neurônios em cada uma delas). Esse modelo tem um comportamento atípico, veja que a precisão sobe em um primeiro momento, mas depois ela decresce até estacionar em 0.7989496. Um possível motivo para isso é o sobreajuste (*overfitting*) dos dados. Note que a taxa de verdadeiros positivos (equação 2.1) nesses modelos é 1, o que significa que todos os cogumelos que eram venenosos foram corretamente classifica-

dos. Porém, a taxa de falsos positivos (equação 2.2) também é 1, mostrando que todos os cogumelos que não eram venenosos foram classificados incorretamente.

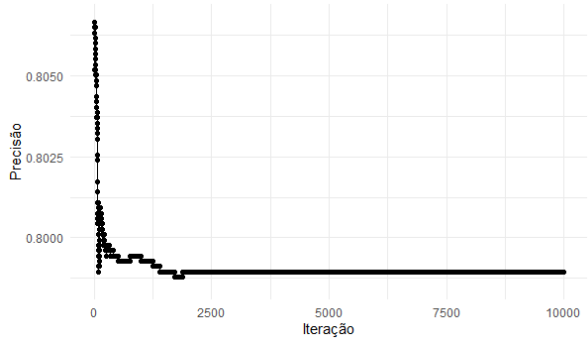

Figura 13: Evolução da precisão no modelo 17

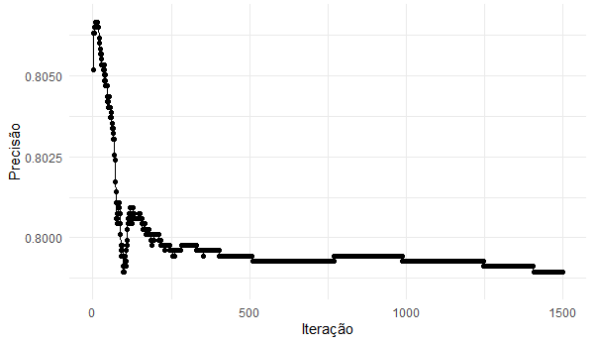

Figura 14: Evolução da precisão no modelo 17 - Primeiras 1500 iterações

Isso mostra que não necessariamente aumentar a quantidade de camadas irá tornar o modelo melhor e também nos lembra a importância de atentar para o sobreajuste.

Concluímos assim que para cada base de dados, uma certa quantidade de camadas e de neurônios se adequará melhor e isso é decidido por validação cruzada. Também que um grande volume de iterações, camadas e ou neurônios não implica na melhora do modelo em relação a precisão do mesmo, de fato é possível que ocasione o efeito contrário produzindo sobreajuste nos dados, além de se tornar cada vez mais custoso computacionalmente.

Referências

DUA, D.; GRAFF, C. *UCI Machine Learning Repository*. 2017. Disponível em: ⟨http://archive.ics.uci.edu/ml⟩.

GOOGLE. *Google Cloud Platform (GCP)*. 2008. Disponível em: ⟨https://cloud.google.com⟩. Acesso em: agosto 2021.

KAPLAN, J. *fastDummies: Fast Creation of Dummy (Binary) Columns and Rows from Categorical Variables*. [S.l.], 2020. R package version 1.6.3. Disponível em: ⟨https://CRAN.R-project.org/package=fastDummies⟩.

KNOPF, A. The audubon society field guide to north american mushrooms. *GH Lincoff (Pres.), New York*, 1981.

KUHN, M. *caret: Classification and Regression Training*. [S.l.], 2021. R package version 6.0-88. Disponível em: ⟨https://CRAN.R-project.org/package=caret⟩.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943.

MELLO, R. F.; PONTI, M. A. *Machine learning: a practical approach on the statistical learning theory*. [S.l.]: Springer, 2018.

MONDELLI, M. L.; PETERSON, A. T.; GADELHA, L. M. Exploring reproducibility and fair principles in data science using ecological niche modeling as a case study. In: SPRINGER. *International Conference on Conceptual Modeling*. [S.l.], 2019. p. 23–33.

PINHEIRO, M. *PerceptronMulticamadas*. 2021. Disponível em: ⟨https://github.com/ShadowNig/PerceptronMulticamadas⟩. Acesso em: 05 agosto 2021.

R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2014. Disponível em: ⟨http://www.R-project.org/⟩.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958.

TIERNEY, N. visdat: Visualising whole data frames. *JOSS*, Journal of Open Source Software, v. 2, n. 16, p. 355, 2017. Disponível em: ⟨http://dx.doi.org/10.21105/joss.00355⟩.

WICKHAM, H.; HESTER, J. *readr: Read Rectangular Text Data*. [S.l.], 2020. R package version 1.4.0. Disponível em: ⟨https://CRAN.R-project.org/package=readr⟩.

APÊNDICE 1 – Base de dados

A base ***mushrooms*** do artigo de Knopf (1981) consiste de 22 características refentes a 8124 cogumelos. Ela possui uma coluna de rótulo que indica quais cogumelos são venenosos e quais não são. Ela foi obtida em 18/06/2021 às 16h58min do repositório de aprendizado de máquinas da UCI (DUA; GRAFF, 2017).

Utilizamos o *software R* (R Core Team, 2014) para toda a manipulação desde a leitura das bases até a construção do algoritmo e por fim a geração dos modelos. Para fazer a leitura do arquivo, foi utilizado o pacote *readr* (WICKHAM; HESTER, 2020).

Após a leitura, utilizamos a função *nearZeroVar* do pacote *caret* (KUHN, 2021), para verificar se haviam variáveis onde todas ou quase todas as observações se encaixavam em apenas uma característica. Vimos que a variável *"gill-attachment"* indicava que 97,4% dos cogumelos possuía himênio (parte inferior do chapéu) livre. Da mesma forma, a variável *"veil-color"* indicava que 97,5% dos cogumelos da base tinham véus brancos. Por fim, identificamos a variável *"veil-type"* que indicava que todos os cogumelos da base possuíam véus parcial. Essas três variáveis foram removidas da análise porque não acrescentariam muita informação ao modelo.

Em seguida, utilizamos a função *vis_miss* do pacote *visdat* (TIERNEY, 2017) para visualizar se haviam dados faltantes na base e como estavam distribuídos. Notamos que a variável *"stalk-root"* era a única com falta de dados e estes representavam 30,5% dela. Assim, optamos por retirá-la.

Dessa forma, ficamos com 19 variáveis na base incluindo o rótulo.

Utilizamos a função *createDataPartition* do pacote *caret* (KUHN, 2021) para fazer a divisão da base em uma amostra de treino e outra de teste mantendo a proporção de classe dos rótulos nas duas amostras. Para isso, definimos uma semente, garantindo a reprodutibilidade do processo. Separamos assim 6093 observações para treinar o modelo enquanto 2031 serviram para teste.

Por fim, foi utilizado a função *dummy_cols* do pacote *fastDummies* (KAPLAN, 2020) para transformar as variáveis categóricas em indicadoras. Terminamos com 88 variáveis em cada amostra, incluindo a de rótulo.

Mais detalhes podem ser encontrados no apêndice 3 ou no repositório do *github* preparado para essa pesquisa (PINHEIRO, 2021).

APÊNDICE 2 – O Algoritmo e os modelos

O algoritmo foi baseado no desenvolvido por Mello e Ponti (2018) em seu livro, generalizando a quantidade de camadas ocultas. O script pode ser encontrado no repositório do *github* preparado para essa pesquisa (PINHEIRO, 2021). Como critério de parada, utilizamos 10 000 iterações ou, caso atingisse, 24h de treinamento.

Seguindo a recomendação de Mondelli, Peterson e Gadelha (2019) em seu artigo, buscamos gerar os modelos de forma reprodutível. Desse modo, além de definir uma semente antes de cada geração aleatória, também disponibilizamos no repositório do *github* e no apêndice 3 as especificações dos *softwares* utilizados e versões dos pacotes. Também usamos uma máquina virtual alocada nos servidores da *Google Cloud* (GOOGLE, 2008) para ter medidas de tempo de processamento mais confiáveis independentes da máquina (as especificações da máquina virtual podem ser encontradas no repositório do *github* (PINHEIRO, 2021) e no apêndice 3).

O critério de decisão usado para classificação final dos dados foi o da **curva ROC** (*Receiver Operating Characteristic Curve*).

No nosso modelo, retornamos valores entre 0 e 1 por causa da função de ativação sigmoidal, depois disso precisamos pegar esses valores e decidir a partir de qual ponto de corte eles serão classificados como 1 (possuindo a característica de interesse) ou 0 (não possuindo). Nem sempre utilizar 0,5 é a melhor escolha. O método da curva ROC consiste em ordenar os resultados e entre cada resultado obtido, pegar um ponto de corte e avaliar a **taxa de verdadeiros positivos** (equação 2.1) e a **taxa de falsos positivos** (equação 2.2). Essas taxas nos dão valores entre 0 e 1 e queremos que a primeira seja o maior possível enquanto que a segunda seja pequena.

$$\text{Taxa de Verdadeiros Positivos} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos Negativos}} \tag{2.1}$$

$$\text{Taxa de Falsos Positivos} = \frac{\text{Falsos Positivos}}{\text{Falsos Positivos} + \text{Verdadeiros Negativos}} \tag{2.2}$$

		Real	
		1	**0**
Previsto	**1**	Verdadeiro positivo	Falso positivo
	0	Falso negativo	Verdadeiro negativo

Tabela 2: Matriz de confusão

Por exemplo, vamos dizer que treinamos um modelo e passamos 10 novas observações para serem classificadas. Os resultados ordenados foram (0.02, 0.03, 0.50, 0.55, 0.69, 0.70, 0.81, 0.87, 0.95, 0.97). Os rótulos reais dessas 10 observações são (0, 0, 0, 1, 0, 0, 1, 1, 1, 1) respectivamente. Definimos então pontos de corte como sendo por exemplo o ponto médio entre pares subsequentes. dessa forma, temos o seguinte vetor de pontos de corte (0.025, 0.265, 0.525, 0.620, 0.695, 0.755, 0.840, 0.910, 0.960). Para o primeiro ponto de corte, vamos classificar todos os resultados abaixo de 0.025 como 0 e os resultados acima como 1 ficando com o seguinte resultado.

Real	Predito
0	0
0	1
0	1
1	1
0	1
0	1
1	1
1	1
1	1
1	1

		Real	
		1	**0**
Previsto	**1**	5	4
	0	0	1

Figura 15: Exemplo: ponto de corte 0.025

Assim, podemos calcular as taxas de valores positivos.

$$\text{Taxa de Verdadeiros Positivos} = \frac{5}{5+0} = \frac{5}{5} = 1$$

$$\text{Taxa de Falsos Positivos} = \frac{4}{4+1} = \frac{4}{5} = 0.8$$

Dessa forma, temos o primeiro ponto na nossa curva ROC (0.8, 1). Em seguida, repetimos esses cálculos para os outros pontos de corte e depois plotamos todos em um gráfico assim

como ilustrado na figura 16.

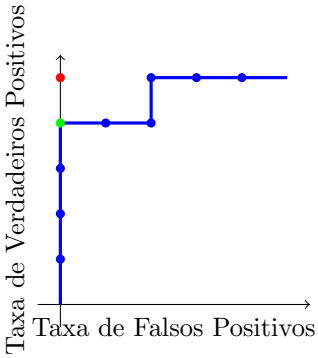

Figura 16: Exemplo: curva ROC

Queremos encontrar o ponto de corte que tenha a melhor TVP e TFP. Para otimizar essa decisão, usamos a distância euclidiana (equação 2.3). para encontrar o ponto de corte que tivesse mais próximo, isto é, com a menor distância, do ponto (0,1) (em vermelho na figura 16). Em outras palavras, com uma alta taxa de verdadeiros positivos e uma baixa taxa de falsos positivos.

$$distância = \sqrt{((0 - TFP)^2) + ((1 - TVP)^2)} \tag{2.3}$$

Determinado o ponto de corte, o utilizamos para classificar as observação como 1 (contendo a característica de interesse) caso o resultado do modelo seja maior que ele ou 0 (não contendo) caso seja menor. No nosso exemplo, o melhor ponto de corte foi o 0.755 que está em verde na figura 16.

Além disso, também calculamos a **área sob a curva** (AUC - *Area Under the Curve*) que nos permite comparar modelos. Na figura 17, plotamos a curva ROC dos modelos 13 e 16 da base *mushrooms*. Nesse exemplo, o modelo 13 possui a maior AUC (98,9%) e portanto é melhor que o modelo 16 (AUC=82,2%).

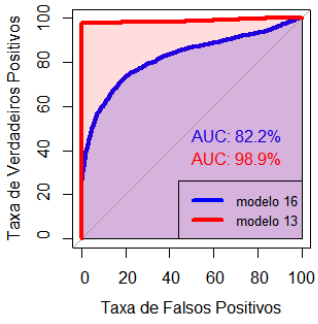

Figura 17: Comparando modelos com AUC

APÊNDICE 3 – Especificações

Seguindo a recomendação de Mondelli, Peterson e Gadelha (2019) em seu artigo, buscamos gerar os modelos de forma reprodutível. As especificações desse apêndice se encontram todas no repositório do *github* (PINHEIRO, 2021).

Os modelos foram rodados em uma máquina virtual do *Google Cloud* (GOOGLE, 2008) com as seguintes características.

- Tipo de máquina: e2-standard-4 (4 vCPUs, 16 GB de memória)
- Zona: us-east4-c
- Ubuntu 16.04.7 LTS (GNU/Linux 4.15.0-1098-gcp x86_64)
- R, versão 4.1.0
- Git, versão 2.0
- Screen, versão 4.03.01

Todos os modelos foram gerados com semente 92021 por meio da função `set.seed()`. Para cada base foi gerado 18 modelos com as seguintes quantidades de neurônios e camadas:

modelo 1 camadas: 1; neurônios: 2;

modelo 2 camadas: 1; neurônios: 3;

modelo 3 camadas: 1; neurônios: 5;

modelo 4 camadas: 2; neurônios: 2 e 2;

modelo 5 camadas: 2; neurônios: 2 e 3;

modelo 6 camadas: 2; neurônios: 2 e 5;

modelo 7 camadas: 2; neurônios: 3 e 2;

modelo 8 camadas: 2; neurônios: 3 e 3;

modelo 9 camadas: 2; neurônios: 3 e 5;

modelo 10 camadas: 2; neurônios: 5 e 2;

modelo 11 camadas: 2; neurônios: 5 e 3;

modelo 12 camadas: 2; neurônios: 5 e 5;

modelo 13 camadas: 5; neurônios: 2 em cada camada;

modelo 14 camadas: 5; neurônios: 3 em cada camada;

modelo 15 camadas: 5; neurônios: 5 em cada camada;

modelo 16 camadas: 10; neurônios: 2 em cada camada;

modelo 17 camadas: 10; neurônios: 3 em cada camada;

modelo 18 camadas: 10; neurônios: 5 em cada camada;

Foi utilizadas a base *mushrooms*, baixada em 18/06/2021 16h58min no repositório de Aprendizado de Máquinas da UCI (DUA; GRAFF, 2017). No pré-processamentos,

- foram removidas as variáveis de variância quase zero ("gill-attachment", "veil-type"e "veil-color"), utilizando a função `caret::nearZeroVar`;
- foi removida a variável "stalk-root"que possui 30,5% de dados faltantes;
- foi utilizada a função `caret::createDataPartition`, para fazer a divisão da base em uma amostra de treino e outra de teste mantendo a proporção de classe dos rótulos nas duas amostras (utilizando a semente 92021);
- por fim, foi utilizado a função `fastDummies::dummy_cols` para transformar as variáveis categóricas em indicadoras.

Para o tratamento e pré-processamento da base de dados foram usados os seguintes pacotes pelo RStudio (versão 1.4.1717)

- `readr`, versão 1.4.0

- `dplyr`, versão 2.1.1
- `caret`, versão 6.0-88
- `fastDummies`, versão 1.6.3
- `visdat`, versão 0.5.3

Observações do algoritmo

- Esse código foi pensado em cima do algoritmo desenvolvido por Mello e Ponti (2018) em seu livro *A Practical Approach on the Statistical Learning Theory*, generalizando a quantidade de camadas ocultas.
- Esse código foi construído para ser utilizado com a função de ativação sigmoidal (2.1). Por esse motivo, as saídas esperadas devem ser entre 0 e 1.
- É necessário que todas as variáveis sejam numéricas. Caso existam variáveis categóricas, essas devem estar no formato de variável indicadora (*dummie*).
- Quando existir mais duas 2 categorias na variável resposta, esta deve estar no formato de variável indicadora (*dummie*).