UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE ESTATÍSTICA

ANÁLISE E IMPLEMENTAÇÃO DE REDES NEURAIS GENERALIZADAS

POR
GUILHERME GUIMARÃES MOREIRA

Dissertação apresentada ao Curso de Mestrado do Departamento de Estatística da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Estatística.

Orientador: Marcelo Azevedo Costa

19 de Dezembro de 2006

AGRADECIMENTOS

Agradeço primeiramente a Deus, por todos os dons que tem me dado ao longo de minha caminhada, pela força e perseverança para superar os obstáculos e por guiar o meu caminho.

Agradeço a meus pais, Lídia e Mateus, e aos meus irmãos, Valéria e Mateus ("Teteu"), pelo apoio, compreensão, carinho e conselhos. À Érika, por sua compreensão e seu amor.

Ao Rafael, ao Carlos e à Talita, meus amigos de Pastoral Universitária, por tudo, mas principalmente pela amizade.

Agradeço a todos os meus colegas de turma, Luciano, Paula, Renata, Fábio, Michele, Lionardo, Mário, Alicia, Carlito e a todos com os quais convivi neste período pela amizade e apoio nas horas difíceis e por compartilharem também as horas alegres.

Agradeço a todos os professores, pelos ensinamentos e conselhos, em particular aos professores Enrico e Sueli, pelas sugestões durante a qualificação e à professora Arminda por me ceder a base de dados de sobrevivência e em especial ao Professor Marcelo, meu orientador, pelo apoio técnico e moral, pela paciência e compreensão nos meus momentos conturbados, e pela orientação com a qual conduziu o trabalho, obrigado!

Aos meus colegas de trabalho da COMEQ, pelo incentivo, principalmente à Ismênia pela ajuda e pelas bases de dados.

Ao IBGE pela licença e pelo suporte financeiro.

A todos aqueles a quem eu não citei, mas que fizeram parte desta jornada, pelo apoio, compreensão, amizade, conselho, risadas, valeu gente!!! Obrigado!!!

À meus pais, Lídia e Mateus

Resumo

Esta dissertação propõe o estudo e a análise de modelos de redes neurais generalizadas. Estes modelos agregam a estrutura de verossimilhança dos modelos lineares generalizados e a flexibilidade das redes neurais artificiais na modelagem de interações não-lineares e não-aditivas entre as variáveis preditoras e a variável resposta. O treinamento é realizado segundo o método interativo do gradiente descendente, que procura minimizar a função desvio do modelo. O critério de qualidade do modelo é obtido via validação cruzada. Os resultados preliminares mostram que as redes neurais generalizadas apresentam resultados de previsão de excelente qualidade quando comparadas com os modelos lineares generalizados, de regressão de Cox e com a rede neural normal.

Palavras Chave: *Redes Neurais, Análise de Sobrevivência, Modelos Lineares Generalizados, Validação Cruzada.*

Sumário

Lista de Figuras

Lista de Tabelas e Quadros

Capítulo 1

Introdução

As Redes Neurais Artificiais (RNAs) têm recebido grande atenção por parte de pesquisadores de diversas áreas, sendo utilizadas nos mais diversos problemas de modelagem de bases de dados. Na estatística, as RNA são cada vez mais utilizadas em problemas de classificação e predição em virtude da sua capacidade de representação interna caracterizada pelo paralelismo inerente à sua arquitetura, possibilitando desempenho de predição superior aos modelos convencionais. Como conseqüência, as RNAs conseguem modelar efeitos não-lineares e não-aditivos das covariáveis em relação à variável resposta como também outras relações existentes entre as próprias covariáveis e que não foram relacionadas *a priori* pelo pesquisador para a confecção de um modelo estatístico convencional.

Dentre os problemas de modelagem estatística, seja classificar um determinado indivíduo em um grupo ou predizer a sua resposta a um determinado tratamento com base em algumas de suas características, os mesmos podem ser modelados segundo a teoria dos Modelos Lineares Generalizados. Estes modelos permitem realizar a modelagem estatística de dados multivariados associando à variável resposta uma distribuição definida na família exponencial. Entretanto, em algumas situações, a escolha de um modelo pode ser um problema complexo devido à falta de informação sobre a variável de interesse e sua correlação com as demais variáveis preditoras e/ou pela existência de muitos fatores não-lineares e não-aditivos a serem estimados (Biganzoli *et al*, 1998). Em tais situações, pode ser mais apropriado considerar modelos flexíveis que sejam capazes de proporcionar uma resposta coerente de predição, seja para um problema de regressão ou classificação, representando internamente as várias correlações existentes e desconhecidas. Neste contexto, as RNAs podem ser

consideradas como modelos flexíveis apropriados para a resolução de problemas multivariados não-lineares. Segundo Biganzolli *et al* (1998), as RNA podem ser consideradas como uma generalização não-linear dos MLGs. Em função de sua característica computacional, a avaliação do potencial da resposta produzida pelas RNAs para predição e/ou classificação deve ser baseada na comparação empírica das redes com outros métodos estatísticos aplicados a dados simulados e reais.

Neste trabalho, deseja-se avaliar a capacidade dos modelos denominados Redes Neurais Generalizadas, que se distinguem dos modelos de RNAs convencionais pela incorporação da função de verossimilhança da família exponencial à estrutura não-linear da RNA. A partir da formulação deste modelo é avaliada a sua capacidade de produzir respostas precisas quando comparadas aos modelos convencionais.

Para tanto, faremos nos capítulos 2, uma revisão sobre os Modelos Lineares Generalizados, no capítulo 3 uma revisão bibliográfica sobre o Modelo de Regressão de Cox, este capítulo foi baseado no livro Análise de Sobrevivência Aplicada de Colosimo e Giolo (2006). No capítulo 4 abordaremos as Redes Neurais Artificiais, faremos um pequeno histórico sobre esta área e seus principais algoritmos. No capítulo 5, explanaremos sobre o novo modelo proposto, a Rede Neural Generalizada. No capítulo 6 faremos comparações empíricas entre as redes MLP *backpropagation* padrão, as RNGs e os MLGs, tanto para banco de dados simulados como para dados reais. No capitulo 7, há uma conclusão do estudo e a apresentação de alguns trabalhos futuros. No Anexo A, estão as rotinas das RNG utilizadas neste estudo.

Capítulo 2

Modelos Lineares Generalizados

Os Modelos Lineares Generalizados (MLG) foram propostos por Nelder e Wedderburn (1972). Esta classe de modelos é baseada na família exponencial uniparamétrica a qual possui propriedades específicas de estimação, teste de hipóteses dentre outras. A idéia básica destes pesquisadores foi a de agregar várias opções para a distribuição da variável resposta, permitindo que a mesma pertencesse a uma única forma paramétrica, definida pela família exponencial de distribuições, bem como dar maior flexibilidade para a relação funcional entre a média da variável resposta e o preditor linear η. Assim, para dados de contagem, onde as ocorrências da variável resposta são independentes, com uma taxa que é função das variáveis explicativas, é de se esperar que a distribuição de Poisson modele bem tais dados. Tal modelo pressupõe que a variância seja proporcional à média e pode ser aplicado para modelar variáveis de contagem em diversos estudos, por exemplo, número de pacientes atendidos diariamente em um pronto socorro. Desta forma ao invés de buscarmos a normalidade dos dados através de uma transformação $\left(x = \sqrt{y}\right)$, podemos admitir que $Y \sim P(\mu)$ e que a relação entre o vetor de médias μ e o preditor linear é dada por: $\log \mu = \eta$, definido em $\Re$.

"Uma importante característica dos Modelos Lineares Generalizados é a suposição de independência entre as observações, ou pelo menos a não correlação entre as mesmas. Com isso podemos concluir que dados auto-correlacionados no tempo (como os utilizados em estudos de séries temporais e econometria) não fazem parte do contexto dos MLGs, bem como modelos nos quais a variável resposta seja modelada por uma distribuição a qual não pertença à família exponencial de distribuições." (Cordeiro e Neto, 2004)

2.1 Definição

Suponha $Y_1, \cdots, Y_n$ variáveis aleatórias independentes, cada uma com densidade expressa da seguinte forma:

$$f_Y(y;\theta,\phi) = \exp\left\{\frac{[y\theta - b(\theta)]}{a(\phi)} + c(y,\phi)\right\} \qquad (2.1)$$

onde,

$a(\cdot), b(\cdot)$ e $c(\cdot)$ são funções conhecidas;

$\phi > 0$ é denominado parâmetro de dispersão e

θ é denominado parâmetro canônico o qual caracteriza a distribuição em (2.1).

Se ϕ é conhecido, a Equação (2.1) representa a família exponencial uniparamétrica indexada por θ.

Os modelos lineares generalizados são definidos pela Equação (2.1) e pela componente sistemática

$$g(\mu_i) = \eta_i \qquad (2.2)$$

onde,

$\eta_i = x_i^T \beta$ é o preditor linear;

$\beta = (\beta_1, \cdots, \beta_p)^T$, $p < n$ é o vetor de parâmetros desconhecidos a serem estimados;

$x_i = (x_{i1}, \cdots, x_{ip})^T$ são os valores das p variáveis explicativas para o i-ésimo indivíduo e

$g(\cdot)$ é a função de ligação, que deve ser monótona e diferenciável.

2.2 Casos Particulares

a) NORMAL

Seja *Y* uma variável aleatória com distribuição normal com média μ e variância σ^2, $Y \sim N(\mu, \sigma^2)$. A densidade de *Y* é dada por:

$$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(y-\mu)^2\right\} = \exp\left\{\frac{y\mu - \mu^2/2}{\sigma^2} - \frac{1}{2}\left(\frac{y^2}{\sigma^2} + \log(2\pi\sigma^2)\right)\right\}$$

onde, $-\infty < y, \mu < \infty$, $\sigma^2 > 0$, $\theta = \mu$, $b(\theta) = \frac{\theta^2}{2}$, $a(\phi) = \sigma^2$ e $c(y,\phi) = \frac{1}{2}\left(\frac{y^2}{\sigma^2} + \log(2\pi\sigma^2)\right)$

b) BINOMIAL

Seja Y^* a proporção de sucessos em *n* ensaios independentes, cada um com probabilidade de ocorrência *p*. Assumiremos que $nY^* \sim B(n, p)$. A densidade de Y^* é dada por:

$$\binom{n}{ny^*} p^{ny^*}(1-p)^{n-ny^*} = \exp\left\{\log\binom{n}{ny^*} + ny^* \log\left(\frac{p}{1-p}\right) + n\log(1-p)\right\}$$

onde, $0 < p, y^* < 1$, $\theta = \log\left(\frac{p}{1-p}\right)$, $b(\theta) = \log(1+e^{\theta})$, $a(\phi) = \frac{1}{n}$ e $c(y^*, \phi) = \log\binom{n}{ny^*}$

O modelo Binomial pode ser utilizado em um número muito grande de situações, as quais apresentam as observações da variável resposta nY^*, como sendo contagens não negativas limitadas a um valor fixo. Um exemplo de utilização do modelo Binomial é o caso onde no qual se deseja descobrir se o domicílio *i* utiliza os serviços telefônicos de uma determinada empresa telefônica de ligações de longa distância ou não.

c) GAMA

Caso $Y \sim \Gamma(\mu, \nu)$ a densidade de *Y* é dada por:

$$\frac{1}{\Gamma(\nu)}\left(\frac{\nu y}{\mu}\right)^{\nu} \exp\left(-\frac{\nu y}{\mu}\right) d(\log y) = \exp\left[\nu\left\{-\frac{y}{\mu} + \log\left(\frac{1}{\mu}\right)\right\} + (\nu\log(\nu y) - \log y - \log\Gamma(\nu))\right]$$

onde, $\nu, \mu > 0$, $y \geq 0$, $\Gamma(\nu) = \int_0^{\infty} t^{\nu-1}e^{-t}dt$ é a função gama, $\theta = -\frac{1}{\mu}$, $b(\theta) = -\log(-\theta)$, $a(\phi) = \nu^{-1}$ e $c(y, \phi) = \nu\log(\nu y) - \log y - \log\Gamma(\nu)$.

Para $0 < \nu < 1$ a densidade da gama atinge seu máximo na origem e decresce monotonicamente quando $y \to \infty$. Quando $\nu = 1$, temos a exponencial, que é um caso particular da gama. Para $\nu > 1$, a densidade assume zero na origem, tem um máximo em $y = \mu - \mu/\nu$ e depois decresce quando $y \to \infty$. Outro caso especial do modelo gama é a Qui-quadrado, a gama assume esta forma quando $\nu = k/2$ e $\mu = k$. Fazendo $\nu \to \infty$ temos a distribuição normal.

d) POISSON

Caso $Y \sim P(\lambda)$ a densidade de *Y* é dada por:

$$\frac{e^{-\lambda}\lambda^{y}}{y!} = \exp\{y\log(\lambda) - \lambda - \log(y!)\},$$

onde, $\lambda > 0$, $y = 0,1,2,\cdots$, $\theta = \log(\lambda)$, $b(\theta) = \theta^2$, $a(\phi) = 1$, $c(y,\phi) = \log(y!)$.

O Quadro 2.1 apresenta um resumo das distribuições mais importantes sob a forma da família exponencial e algumas de suas características principais:

Quadro 2.1: *Principais distribuições pertencentes à família exponencial de distribuições*

Modelo	$a(\phi)$	$b(\theta)$	θ	$V(\mu)$
$N(\mu,\sigma^2)$	σ^2	$\frac{\theta^2}{2}$	μ	1
$P(\lambda)$	1	e^{θ}	$\log\lambda$	λ
$\frac{B(n,p)}{n}$	$\frac{1}{n}$	$\log(1+e^{\theta})$	$\log\left\{\frac{p}{1-p}\right\}$	$p(1-p)$
$\Gamma(\mu,\upsilon)$	υ^{-1}	$-\log(-\theta)$	$-\frac{1}{\mu}$	μ^2
$N^{-}(\mu,\phi)$	ϕ	$-\sqrt{-2\theta}$	$-\frac{1}{\mu^2}$	μ^3

2.3 Função de Ligação

A função de ligação relaciona o preditor linear η ao vetor de médias μ da variável resposta Y. Considere a estrutura linear de um modelo de regressão:

$$\eta = X\beta$$

onde, $\eta = [\eta_1,\ldots,\eta_n]^T$, $\beta = [\beta_1,\ldots,\beta_n]^T$ e X é uma matriz *n* por *p*, com *n*>*p*, conhecida de posto *p*.

A função linear η dos parâmetros β, desconhecidos, é chamada *preditor linear*. Outra característica da componente sistemática (X) é que o vetor de médias μ da

variável resposta *Y* pode ser expressa por uma função conhecida monótona e diferenciável de η_0 denominada função de ligação:

$$\mu_i = g^{-1}(\eta_i), \text{ i = 1,2,...,n}$$

Para o caso de *Y*~*P*(μ), μ>0, uma função de ligação adequada seria a função logarítmica ($\eta = \log \mu$), pois possui domínio positivo e seu contradomínio é o conjunto de números reais.

2.3.1 Função de Ligação Canônica

Seja $Y = [y_1, \cdots, y_n]^T$ um vetor aleatório e seja $f_{y_1}(y_1, \theta_1, \phi), \cdots, f_{y_n}(y_n, \theta_n, \phi)$ as respectivas funções de densidade das variáveis aleatórias pertencentes ao vetor *Y*, definidas pela Equação (2.1). A função de verossimilhança de *Y* é dada por:

$$L(\theta, \phi, y) = \prod_{i=1}^{n} f_{y_i}(y_i, \theta_i, \phi) = \prod_{i=1}^{n} \exp\left\{ \frac{[y_i \theta_i - b(\theta_i)]}{a(\phi)} + c(y_i, \phi) \right\} \qquad (2.3)$$

A log-verossimilhança, logaritmo da função de verossimilhança, de um MLG pode ser expressa como sendo:

$$l(\theta, \phi; y) = \sum_{i=1}^{n} \frac{1}{a(\phi)} \{ y_i \theta_i - b(\theta_i) \} + \sum_{i=1}^{n} c(y_i, \phi) \qquad (2.4)$$

Um caso particular importante das funções de ligação ocorre quando o parâmetro canônico (θ) coincide com o preditor linear, isto é, quando $\theta_i = \eta_i = \sum_{j=1}^{p} x_{ij} \beta_j$. Nesse caso, $l(\theta, \phi; y) = l(\beta, \phi; y)$ é dado por:

$$l(\beta, \phi; y) = \sum_{i=1}^{n} \frac{1}{a(\phi)} \left\{ y_i \sum_{j=1}^{p} x_{ij} \beta_j - b\left(\sum_{j=1}^{p} x_{ij} \beta_j \right) \right\} + \sum_{i=1}^{n} c(y_i, \phi)$$

Definindo a estatística $S_j = \frac{1}{a(\phi)} \sum_{i=1}^{n} Y_i x_{ij}$, $l(\beta, \phi, y)$ fica então expresso na forma:

$$l(\beta, \phi; y) = \sum_{j=1}^{p} s_j \beta_j - \frac{1}{a(\phi)} \sum_{i=1}^{n} b\left(\sum_{j=1}^{p} x_{ij} \beta_j \right) + \sum_{i=1}^{n} c(y_i, \phi)$$

Logo, pelo teorema da fatorização a estatística $S = [S_1, \ldots, S_n]^T$ é suficiente minimal para o vetor $\beta = [\beta_1, \ldots, \beta_n]^T$. As ligações que correspondem a tais estatísticas são chamadas de ligações canônicas. As ligações canônicas para os modelos: normal, binomial, poisson e gama são, respectivamente, dadas por:

$$\mu = \eta,\ \log\left\{\frac{\mu}{1-\mu}\right\} = \eta,\ \log\mu = \eta \text{ e } \mu^{-1} = \eta.$$

Uma grande vantagem em se utilizar as ligações canônicas é que as mesmas garantem a concavidade de $l(\beta, \phi; y)$ facilitando desta forma a obtenção de muitos resultados assintóticos.

2.3.2 Outras funções de ligação

a) LIGAÇÃO PROBITO

Seja *p* a proporção de sucessos de uma distribuição binomial. A ligação probito é definida por:

$$\Phi(p) = \eta,$$

onde, $\Phi(\cdot)$ é a função de distribuição acumulada da normal padrão.

b) LIGAÇÃO COMPLEMENTO LOG-LOG

A distribuição do valor extremo (logaritmo da exponencial) tem densidade dada por:

$$f(y) = \exp\{y - e^y\}, \quad -\infty < y < \infty$$

Logo, a função de distribuição acumulada fica dada por:

$$F(y) = 1 - \exp\{-e^y\}.$$

O modelo binomial com ligação complemento log-log é definido tal que

$$\mu = 1 - \exp\{-e^{\eta}\}.$$

ou, equivalentemente,

$$\log(-\log(1-\mu)) = \eta.$$

2.4 Adequação do Modelo

Formulado o modelo, torna-se necessário estimar os parâmetros e avaliar a precisão das estimativas. As estimativas dos parâmetros são realizadas via o algoritmo de estimação denominado M*étodo Escore de Fisher* (Nelder e Wedderburn, 1972), o qual é baseado no algoritmo de Newton-Raphson.

Nos MLGs, o processo de estimação é determinado por uma medida de bondade do ajuste entre os dados observados e os valores estimados a partir do mesmo. As estimativas dos parâmetros do modelo serão aquelas que minimizam esta medida, ou seja, aquelas que maximizam da log-verossimilhança. Logo as estimativas dos parâmetros podem ser obtidas através da maximização da log-verossimilhança em relação aos parâmetros. Se $f_Y(y;\theta,\phi)$ é a função de densidade para a observação y, dado o parâmetro θ e supondo ϕ conhecido. Então a log-verossimilhança expressa como função do valor esperado $\mu = E(Y)$ é dada por:

$$l(\mu, y) = \log f_Y(y;\theta,\phi).$$

onde, $\mu = (\mu_1, \cdots, \mu_n)^T$, $y = (y_1, \cdots, y_n)^T$.

Uma medida da bondade do ajuste conhecida é a função desvio, a qual será abordada na seção 2.4.1.

2.4.1 Função Desvio

Uma das muitas medidas de discrepância ou bondade do ajuste existentes é a função desvio, a qual equivale à diferença de log verossimilhanças maximizadas.

Sabe-se, que podemos construir para uma amostra de *n* elementos modelos com até *n* parâmetros. O mais simples deles, denominado *modelo nulo*, contém apenas 1 parâmetro, o qual representa a média μ, é muito simples. Por outro lado, o *modelo saturado,* o qual contém n parâmetros, um para cada observação, é não informativo, porém é útil para medir a discrepância de um modelo intermediário com *p* parâmetros ($p<n$).

Seja $y=(y_1,\cdots,y_n)^T$ uma amostra aleatória com distribuição pertencente à família exponencial, ou seja, com densidade expressa por (2.1). Sejam $\hat{\theta}=\theta(\hat{\mu})$ e $\tilde{\theta}=\theta(y)$ as estimativas dos parâmetros canônicos para o *modelo intermediário*, sob investigação, e o *modelo saturado*, respectivamente.aloca toda a variação da amostra na componente sistemática. Sejam $l(\hat{\theta},\phi;y)$ e $l(\tilde{\theta},\phi;y)$ as respectivas funções de log-verossimilhança para os modelos intermediário e saturado, dadas por:

$$l(\hat{\theta},\phi;y)=\sum_{i=1}^{n} l(\hat{\theta}_i,\phi;y)=\sum_{i=1}^{n}\left\{\frac{y_i\hat{\theta}_i-b(\hat{\theta}_i)}{a_i(\phi)}+c(y_i,\phi)\right\},$$
$$l(\tilde{\theta},\phi;y)=\sum_{i=1}^{n} l(\tilde{\theta}_i,\phi;y)=\sum_{i=1}^{n}\left\{\frac{y_i\tilde{\theta}_i-b(\tilde{\theta}_i)}{a_i(\phi)}+c(y_i,\phi)\right\}.$$

Assumindo $a_i(\phi)=\phi$, caso mais comum, podemos escrever a função desvio como sendo o dobro da diferença entre a log-verossimilhança do *modelo intermediário* para com o *modelo saturado*, temos que:

$$2\left(l(\hat{\theta},\phi;y)-l(\tilde{\theta},\phi;y)\right)=2\sum_{i=1}^{n}\left\{\frac{y_i(\tilde{\theta}_i-\hat{\theta}_i)-b(\tilde{\theta}_i)+b(\hat{\theta}_i)}{\phi}\right\}=\frac{D}{\phi},$$

onde, $D=2\sum_{i=1}^{n} y_i(\tilde{\theta}_i-\hat{\theta}_i)-b(\tilde{\theta}_i)+b(\hat{\theta}_i)$ é denominado *desvio do modelo*, o qual esta sendo investigado. Em geral um modelo com baixo desvio é considerado significativo se $D<\chi^2_{n-p}$, onde *n* é o tamanho da amostra e *p* o número de parâmetros a serem estimados.

A seguir apresentamos as formas da função desvio para as principais distribuições da família exponencial.

Normal $$D=\sum_{i}^{n}(y_i-\hat{\mu}_i)^2 \qquad (2.5)$$

Binomial $$D=2\sum_{i=1}^{n}\left[y_i\log\left(\frac{y_i}{\hat{\mu}_i}\right)+(n_i-y_i)\log\left(\frac{n_i-y_i}{n_i-\hat{\mu}_i}\right)\right] \qquad (2.6)$$

Gama $$D=2\sum_{i=1}^{n}\left[\log\left(\frac{\hat{\mu}_i}{y_i}\right)+\frac{y_i-\hat{\mu}_i}{\hat{\mu}_i}\right] \qquad (2.7)$$

Poisson $$D=2\sum_{i=1}^{n}\left[y_i\log\left(\frac{y_i}{\hat{\mu}_i}\right)-(y_i-\hat{\mu}_i)\right] \qquad (2.8)$$

Pode-se notar que a função desvio para o modelo Normal, expressa pela Equação (2.5), é a soma dos quadrados dos resíduos.

Para finalizar gostaria de destacar as vantagens e desvantagens dos MLGs. A sua principal vantagem é que a variável resposta da regressão pode não ser regida pela distribuição Normal, ela pode ser regida por qualquer lei que pertença à família de distribuições uniparamétrica conhecida como *família exponencial*. Outra vantagem é a de que os seguintes modelos: Modelo de Regressão Normal, Modelos Log-lineares (aplicados a tabelas de contingência), Modelos Logísticos (aplicados a tabelas multidimensionais de proporções), Modelos Estruturais (com erro gama), entre outros, são casos particulares dos MLGs. Entretanto, os MLG não englobam os modelos para dados correlacionados, como os utilizados em séries temporais e econometria, e aqueles onde a distribuição da variável resposta não pertença à *família exponencial*. Porém, alguns casos de regressão que originalmente não fazem parte do contexto dos MLGs podem ser ajustados através de algoritmos iterativos, mediante algumas alterações (Cordeiro e Paula, 1992).

Capítulo 3

Modelo de Regressão de Cox

O objetivo deste capítulo é apresentar o modelo de Cox para a análise de sobrevivência. Este importante modelo estatístico nos permite analisar dados provenientes de estudos de tempos de vida nos quais a resposta é o tempo até a ocorrência do evento de interesse.

O modelo de Cox é um modelo estatístico popular. Uma das principais razões para tal popularidade é a presença, no modelo, de uma componente não-paramétrica, o que o torna bastante flexível. Outra razão de sua popularidade é o fato relatado por Kalbfleisch e Prentice (1980), onde o modelo de Cox apresenta alguns modelos paramétricos como casos particulares seus, como por exemplo o modelo de Weibull e o modelo exponencial (que é um caso particular do modelo de Weibull).

Para uma melhor compreensão do mesmo, iniciaremos com um modelo simples com apenas uma covariável a qual é o grupo de tratamento para logo em seguida apresentá-lo em sua forma mais geral.

Suponha um estudo controlado onde os n indivíduos que participam dele foram divididos aleatoriamente em 2 grupos de tratamento (grupos 0 e 1), os indivíduos do grupo 0 receberam o tratamento padrão enquanto que os do outro grupo um tratamento alternativo. Sejam $\lambda_0(t)$ e $\lambda_1(t)$ as respectivas funções de taxa de falha dos grupos 0 e 1 e, assumindo que proporcionalidade entre elas ao longo do tempo seja constante, ou seja, independa do tempo temos, que:

$$\frac{\lambda_0(t)}{\lambda_1(t)} = K$$

onde,

K é a razão entre as taxas de falhas.

Sendo *x* a variável indicadora de grupo, tal que:

$$x = \begin{cases} 0, & \textit{se o indivíduo recebeu o tratamento padrão} \\ 1, & \textit{se o indivíduo recebeu o tratamento alternativo} \end{cases}$$

e $K = \exp\{\beta \cdot x\}$, então temos que,

$$\lambda(t) = \lambda_0(t)\exp(\beta \cdot x), \tag{3.1}$$

ou seja,

$$\lambda(t) = \begin{cases} \lambda_1(t) = \lambda_0(t)\exp(\beta \cdot x), & x = 1 \\ \lambda_0(t), & x = 0 \end{cases}$$

A forma do modelo de Cox para uma única covariável é a definida pela Equação (3.1).

Seja agora para o modelo com p covariáveis. Suponha $x = [x_1, \cdots, x_p]^T$ um vetor p-dimensional, onde cada uma das suas *p* componentes representa uma covariável do modelo. A forma geral do modelo de Cox é:

$$\lambda(t) = \lambda_0(t)g(x^T\beta), \tag{3.2}$$

onde,

g é uma função não-negativa, tal que *g*(**0**) = 1 e

$\beta = (\beta_1, \cdots, \beta_p)$ é o vetor de parâmetros associados às covariáveis do modelo.

Este modelo é composto por duas componentes, uma paramétrica e a outra não-paramétrica. A componente não-paramétrica, $\lambda_0(t)$, é uma função não-negativa do tempo e é denominada função de base, pois quando **x** = **0**, $\lambda(t) = \lambda_0(t)$. Já a componente paramétrica, $g(x^T\beta)$, é freqüentemente utilizada na seguinte forma:

$$g(x^T\beta) = \exp(x^T\beta) = \exp(x_1\beta_1 + \cdots + x_p\beta_p) \tag{3.3}$$

Esta forma, denominada forma multiplicativa, garante que a função de taxa de falhas $\lambda(t)$ seja sempre não-negativa. Existem outras formas para a função $g(x'\beta)$ (Storer et al., 1983), porém a forma multiplicativa é a mais utilizada.

O modelo de Cox para dados de sobrevivência também é denominado de modelo de riscos proporcionais, pois a sua suposição básica é que a razão entre as taxas de falhas entre dois indivíduos *i* e *j* distintos quaisquer é constante no tempo. Isto é, a razão entre as funções de taxa de falhas $\frac{\lambda_i(t)}{\lambda_j(t)} = \exp\left(x_i'\beta - x_j'\beta\right)$, independe do tempo.

3.1 Ajustando o Modelo de Regressão de Cox

O modelo de Cox é caracterizado pelos coeficientes β's, que medem os efeitos das covariáveis. Logo, um método de estimação se faz necessário para que se possa fazer inferências acerca dos parâmetros do modelo. O método de máxima verossimilhança, proposto por Cox e Hinkley (1974) é bastante utilizado para este propósito. No entanto a presença da componente não-paramétrica, $\lambda_0(t)$, torna este método inapropriado, pois a função de verossimilhança, $L(\beta)$, seria função do componente não-paramétrico, como se pode notar na seguinte expressão:

$$L(\beta) = \prod_{i=1}^{n} \left[\lambda_0(t_i)\exp\left(x_i'\beta\right)\right]^{\delta_i} \exp\left\{\int_0^{t_i} \lambda_0(u)\exp\left(x_i'\beta\right)du\right\}$$

onde,

δ_i é a i-ésima ocorrência da variável indicadora de falha $\delta = \begin{cases} 1, & falha \\ 0, & c.c. \end{cases}$

Uma solução razoável consiste em condicionar a construção de $L(\beta)$ ao conhecimento do tempo passado de forma a eliminar a perturbação causada pela componente não-paramétrica. Então, em 1975, Cox formalizou o que havia proposto em seu artigo original e daí surgiu o método de máxima verossimilhança parcial (Cox, 1975).

3.1.1 Método da Verossimilhança Parcial

Considere uma amostra de *n* indivíduos, onde existam $k \leq n$ falhas distintas nos tempos $t_1 < t_2 < \cdots < t_k$, de forma simples a verossimilhança parcial considera que a probabilidade de o i-ésimo indivíduo vir a apresentar o evento de interesse no tempo t_i conhecendo todo o histórico de aparecimento de tal evento até o tempo t_i, é:

P[do indivíduo vir a apresentar evento de interesse em t_i |
Houve um evento de interesse em t_i e todo o histórico sobre este evento até t_i] =

$$\frac{\lambda_i(t \mid x_i)}{\sum_{j \in C_i} \lambda_j(t \mid x_j)} = \frac{\lambda_0(t)\exp(x_i'\beta)}{\sum_{j \in C_i} \lambda_0(t)\exp(x_j'\beta)} = \frac{\exp(x_i'\beta)}{\sum_{j \in C_i} \exp(x_j'\beta)} \qquad (3.4)$$

onde,

C_i é o conjunto de índices de todas as observações com risco de apresentarem o evento de interesse até o tempo até o tempo t_i.

Podemos notar em (3.4) que condicionado ao seu histórico a componente não-paramétrica desaparece. A função de verossimilhança é formada pelo produtório de todos os termos representados por (3.4), ou seja,

$$L(\beta) = \prod_{i=1}^{k} \left(\frac{\exp(x_i'\beta)}{\sum_{j \in C_i} \exp(x_j'\beta)} \right) = \prod_{i=1}^{n} \left(\frac{\exp(x_i'\beta)}{\sum_{j \in C_i} \exp(x_j'\beta)} \right)^{\delta_i} \qquad (3.5)$$

onde,

δ_i é uma variável indicadora de falha (ocorrência evento de interesse).

Os valores de β que maximizam $L(\beta)$ são obtidos resolvendo-se o sistema de equações definido por $U(\beta) = \left(\frac{dl(\beta)}{\partial \beta_1} = 0, \cdots, \frac{dl(\beta)}{\partial \beta_p} = 0 \right)$, onde $l(\beta) = \log(L(\beta))$.

A função de verossimilhança parcial (3.5) assume que os tempos são contínuos, e conseqüentemente, não pressupõe empates. Na prática eles podem ocorrer, pois os tempos na maioria das vezes não são medidos em segundos, sendo muito comum

mensurá-los em dias, meses e até mesmo anos. Porém ela pode ser modificada para incorporar tais acontecimentos (Breslow, 1972 e Peto,1972; Efron, 1977 entre outros).

As propriedades assintóticas dos estimadores de máxima verossimilhança parcial foram demonstradas por diversos autores, porém foram Andersen e Gil (1982) que apresentaram as formas mais gerais destas propriedades. Eles mostraram que estes estimadores são consistentes e assintoticamente normais sob certas condições de regularidade, de forma que é possível utilizarmos as conhecidas estatísticas de Wald, da Razão de Verossimilhanças e Escore para fazermos inferências sobre os parâmetros.

3.2 Interpretação dos Coeficientes do Modelo de Cox

Podemos observar pela expressão (3.2) que as covariáveis possuem o efeito de aumentar ou diminuir a função de risco. No entanto para se interpretar corretamente os coeficientes estimados devemos levar em consideração a propriedade de riscos proporcionais do modelo. Desta forma, razão entre as taxas de falhas de dois indivíduos distintos, *i* e *j*, que têm para todas as covariáveis, exceto para uma, os mesmos valores é:

$$\frac{\lambda_i(t)}{\lambda_j(t)} = \exp\left[\beta_k\left(x_{ik} - x_{jk}\right)\right]$$

onde,

β_k coeficiente estimado para a covariável que apresenta valor distinto para os indivíduos *i* e *j* e

x_{ik}, x_{jk} são os valores da k-ésima covariável para os indivíduos *i* e *j*.

Esta razão é constante para todo tempo, por causa da propriedade de riscos proporcionais do modelo de Cox, ou seja, ela independe da variável tempo. Por exemplo, seja *x* uma covariável dicotômica indicando se o indivíduo é ou não fumante. O risco de aparecimento de câncer de pulmão entre indivíduos que fumam é $\exp(\beta_k)$ vezes o risco entre os que não fumam.

O cálculo da estimativa pontual da razão de falhas, $\exp(\beta_k)$, pode ser realizado utilizando a propriedade de invariância do estimador de máxima verossimilhança parcial, enquanto que para o cálculo de uma estimativa intervalar para a mesma faz-se necessária a obtenção de uma estimativa da variabilidade de $\exp(\beta_k)$, o que pode ser feito com a utilização do *método delta*. Caso o valor 1 pertencesse ao intervalo de

confiança estimado indicaria que o risco de aparecimento de câncer de pulmão e o hábito de fumar não apresentam diferenças significativas.

A mesma idéia pode ser aplicada a variáveis qualitativas as quais dividem a amostra em 3 ou mais grupos. Por exemplo, considere uma covariável de grupo com 3 níveis (0,1 e 2). Esta covariável pode ser representada por 2 variáveis indicadoras, ou seja, os termos referentes à covariável no modelo são $\beta_{k1}x_{k1} + \beta_{k2}x_{k2}$, onde x_{k1} e x_{k2}, são respectivamente, as variáveis indicadores de indivíduo pertencer ao grupo 1 ou 2. Caso ambas sejam 0 isto indica que o indivíduo pertence ao grupo de controle, neste caso o grupo 0. Porém, neste caso, os grupos 1 e 2, são comparados com o grupo de controle, grupo 0, e não entre si.

Para variáveis contínuas a interpretação é bastante similar. Por exemplo, se o efeito de uma variável contínua β_k for significativo, ou seja, $\exp(\hat{\beta}_k)$=1,1; temos que para cada incremento de uma unidade nesta variável o risco de o evento de interesse vir a acontecer fica aumentado em 10%.

3.3 Adequação do Modelo de Cox

Por conta da presença da componente não-paramétrica, o modelo de Cox é bastante flexível. Porém mesmo contando com a presença desta importante componente ele não se ajusta a qualquer situação e, como todo modelo estatístico, requer o uso de técnicas para avaliar a sua adequação. Como mencionado anteriormente o modelo de Cox tem como suposição básica a proporcionalidade dos riscos. Segundo Struthers e Kalbfleisch (1986) a violação desta suposição pode acarretar sérios vícios na estimação dos coeficientes do modelo.

Existem vários métodos para avaliar a adequação do modelo. Eles baseiam-se nos mesmos tipos de resíduos definidos para os modelos paramétricos. Alguns deles são utilizados para verificar a qualidade geral do ajuste do modelo, enquanto que outros em relação à suposição da proporcionalidade dos riscos.

3.3.1 Avaliação da Qualidade Geral do Ajuste do Modelo de Cox

Uma forma de se avaliar a qualidade geral de ajuste do modelo de Cox é utilizando os resíduos de Cox-Snell (1968). Para o modelo de Cox, estes resíduos são definidos como sendo:

$$\hat{e}_i = \hat{\Lambda}_0(t_i)\exp\left(\sum_{k=1}^{p} x_{ip}\hat{\beta}_k\right), \quad i = 1, \cdots, n \tag{3.6}$$

onde,

$\hat{\Lambda}_0(t_i)$ é a estimativa da função de taxa de falha acumulada e pode ser calculada como sendo:

$$\hat{\Lambda}_0(t_i) = \sum_{j:t_j<t} \frac{d_j}{\sum_{k\in C_j} \exp\left(x_k^{'}\hat{\beta}\right)}, \tag{3.7}$$

onde,

d_j é o número de falhas em t_j e

C_j é o conjunto de índices de todas as observações com risco de apresentarem o evento de interesse até o tempo até o tempo t_j.

De forma que se o modelo estiver bem ajustado os resíduos devem ser vistos como uma amostra censurada de uma distribuição exponencial padrão, ou seja, o gráfico $\hat{\Lambda}_0(\hat{e}_i)$ vs. $\hat{e}_i$ deve ser aproximadamente uma reta. Embora tais resíduos sejam úteis para examinar o ajuste global do modelo eles não indicam o tipo de falha detectado, como por exemplo, um comportamento não linear. Sendo assim tais resíduos não são indicados para a suposição de riscos proporcionais.

3.3.2 Avaliação da Suposição da Proporcionalidade dos Riscos

Para avaliar a suposição de riscos proporcionais no modelo de Cox, algumas técnicas gráficas e testes estatísticos encontram-se propostos na literatura, podemos citar o método do gráfico descritivo (o qual será abordado mais adiante), e o método com variável dependente no tempo (Cox,1979) que consiste em acrescentar ao modelo uma covariável dependente no tempo.

Porém, muitos deles têm sérias limitações. Os testes de Shoenfeld (1980) e Andersen (1982) consideram uma partição arbitrária do eixo do tempo para a sua aplicação, o problema aqui é que partições distintas geram testes distintos. O teste de Wei (1984), por exemplo, não necessita desta partição, porém só pode ser utilizado em modelos com uma única covariável.

Devido a todas estas limitações envolvendo os testes de adequação é que as técnicas gráficas envolvendo resíduos definidos adequadamente são ferramentas úteis para tal finalidade. Sendo assim apresentaremos a seguir o método do gráfico descritivo.

Método do Gráfico Descritivo

Esta técnica consiste em dividirmos os dados em *m* estratos de acordo com alguma covariável, por exemplo, em um extrato estão os fumantes e em outro os não-fumantes. Em seguida, estima-se $\hat{\Lambda}_{0_j}(t)$, para cada estrato, a partir de (3.7). Se a suposição de riscos proporcionais for válida, então, as curvas $\log\left[\hat{\Lambda}_{0_j}(t)\right]$ vs. t ou log(t) devem apresentar variações aproximadamente constantes. Curvas não paralelas significam que existe uma quebra da suposição da proporcionalidade dos riscos, por isso é recomendável que se construa este gráfico para todas as covariáveis envolvidas. No caso de covariável contínua a sugestão é: agrupa-la em um pequeno número de categorias.

A grande vantagem desta técnica é que com ela podemos identificar qual covariável viola a suposição de riscos proporcionais. Porém, como toda técnica gráfica, a sua conclusão é subjetiva, pois depende da interpretação dos gráficos.

3.4 Conclusão do Capítulo

Vimos neste capítulo que o modelo de regressão de Cox é um modelo estatístico para a análise de sobrevivência bastante popular, o que é devido principalmente à presença da componente não-paramétrica que o torna bastante flexível. Foi apresentado, respectivamente, nas seções 3.1 à 3.3: o ajuste do modelo via método da verossimilhança parcial, a interpretação dos coeficientes e a adequação do mesmo em relação à sua suposição básica de riscos proporcionais.

Capítulo 4

Redes Neurais Artificiais

As RNAs são sistemas paralelos distribuídos compostos por unidades de processamento simples (nodos ou neurônios) que calculam determinadas funções matemáticas. Tais unidades estão dispostas em uma ou mais camadas e interligadas. Na maioria dos modelos estas conexões estão associadas a pesos, os quais armazenam o conhecimento representado ao modelo e servem para ponderar a entrada recebida por cada nodo da rede.

A solução de problemas via RNAs é bastante atrativa, já que a forma como estes são representados internamente pela rede e o paralelismo natural inerente à sua arquitetura criam a possibilidade de um desempenho superior ao dos modelos convencionais, pois ela consegue estimar tanto fatores lineares quanto não-lineares muito bem, o que aumenta o seu poder de predição e classificação frente aos demais modelos.

Como exemplo, suponhamos duas variáveis de entrada, x_1 e x_2. Nos modelos de regressão linear clássicos, uma forma de tornar o modelo mais flexível consiste em incluir combinações não-lineares fixas entre os regressores, tais como x_1^2, x_2^2, $x_1 x_2$, $x_1^2 x_2$, ... Entretanto, esta aproximação adiciona exponencialmente muitos termos à regressão a medida que a ordem polinomial aumenta.

Em contrapartida, considere um único nodo existente na camada escondida de uma RNA, conectada a 2 entradas. Os parâmetros da rede ajustáveis são α_0, α_1 e α_2. Uma função típica para este nodo é dada pela função *tangente hiperbólica*:

$$\tanh(\alpha_0 + \alpha_1 x_1 + \alpha_2 x_2).$$

executando uma expansão da série de Taylor em y para $\tanh(\alpha_0 + y)$, onde $y = \alpha_1 x_1 + \alpha_2 x_2$ e considerando $\beta = \tanh(\alpha_0)$ tem-se:

$$\tanh(\alpha_0 + \alpha_1 x_1 + \alpha_2 x_2) = \beta + (1 - \beta^2)(\alpha_1 x_1 + \alpha_2 x_2) + (-\beta + \beta^3)(\alpha_1 x_1 + \alpha_2 x_2)^2 + \left(-\frac{1}{3} + \frac{4\beta^2}{3} - \beta^4\right)(\alpha_1 x_1 + \alpha_2 x_2)^3 + \left(\frac{2\beta}{3} - \frac{5\beta^2}{3} + \beta^5\right)(\alpha_1 x_1 + \alpha_2 x_2)^4 + O(\alpha_1 x_1 + \alpha_2 x_2)^5$$

Apesar do número de termos ser infinito, a função não-linear computada por este único nodo inclui todas as representações das variáveis da entrada, mas elas não podem ser todas controladas independentemente. Os termos dependem apenas dos coeficientes α_0, α_1 e α_2. Dessa forma, adicionar mais nodos à rede RNA, aumenta a flexibilidade da função computada pela rede uma vez que cada nodo conectado permite que modelo capture tantos relacionamentos não-lineares entre as variáveis quanto o número de nodos disponíveis.

O procedimento usual de solução de problemas em RNAs passa inicialmente por uma fase de especificação de topologia: número de nodos e disposição dos mesmos e aprendizagem, na qual um conjunto de padrões ou amostra é apresentado para a rede que, por sua vez, extrai as características necessárias para representar a informação fornecida e posteriormente gerar respostas coerentes para o problema.

Esta capacidade de aprender através de exemplos e de generalizar a informação aprendida é, sem dúvida o principal atrativo à solução de problemas via RNAs. A generalização está associada à capacidade de a rede aprender através de um conjunto reduzido de exemplos e posteriormente, inferir resultados para dados desconhecidos. Não obstante, as redes também são capazes de atuar como mapeadores universais de funções multivariáveis com um custo computacional que cresce linearmente de acordo com o número de variáveis.

4.1 Histórico

As RNAs surgiram a partir do trabalho de Warren McCulloch, um psiquiatra e neuroanatomista que dedicou cerca de 20 anos de trabalho à tentativa de representar um evento do sistema nervoso, e Walter Pitts, um matemático recém graduado, que se juntou a ele em 1942. Eles, em 1943, publicaram o artigo “A Logical

Calculus of the Ideas Immament in Nervous Activity", onde é apresentado o primeiro modelo artificial de um neurônio, denominado nodo MCP. Tal artigo concentrou-se mais em descrever o modelo de neurônio artificial do que em apresentar técnicas de aprendizados.

O primeiro trabalho conhecido de que se tem notícia no qual foi apresentada uma regra de aprendizagem foi apresentado por Donald Hebb em 1949 (Hebb, 1949). A regra de Hebb, como é conhecida, propõe que o aprendizado das redes neurais é conseguido através da variação dos pesos de entrada dos nodos. Mais tarde, em 1960, Widrow e Hoff, sugeriram uma regra para o aprendizado conhecida como *regra delta* (Widrow e Hoff, 1960). Esta regra é baseada no método do gradiente descendente para a minimização do erro na saída de um neurônio com resposta linear.

Em 1958, Frank Rosenblatt desenvolve uma nova abordagem para os problemas de reconhecimento de padrões, o perceptron. Rosenblatt (1958) demonstrou que se fossem acrescidas sinapses ajustáveis às RNAs, com nodos MCP, estas poderia ser treinadas para classificar certos tipos de padrões, desde que linearmente separáveis.

Porém em 1969, o artigo de Minsky e Papert (Minsky e Papert, 1969), chamou a atenção para algumas tarefas que o *perceptron* não era capaz de executar. Seu principal argumento era que o problema do crescimento explosivo de espaço ocupado e requerido para a solução de problemas mais complexos, afetaria as redes, inclusive o *perceptron.*. As duras críticas de Minsky e Papert levou a um grande desinteresse na área.

Porém, em 1982, com a publicação do trabalho de John Hopfield (Hopfield, 1982), no qual ele chama a atenção para as propriedades associativas das redes, e posteriormente, com a formulação do algoritmo *backpropagation* (Rumelhart, Hinton e Williams, 1986) é que as pesquisas em RNAs vieram novamente a se intensificar.

A partir da formulação do algoritmo *backpropagation* padrão e em virtude do método do gradiente descendente apresentar uma convergência lenta, foram desenvolvidas variações do método como o *RProp* (Riedmiller e Braun, 1983), *Momentum* (Rumelhart et al., 1986), *QuickProp* (Fahlman,1988), entre outros com o objetivo de melhorar a velocidade de convergência e de impedir a parada do processo em função dos mínimos locais. Apesar dos métodos desenvolvidos apresentarem um menor tempo até a convergência eles não podiam evitar as regiões de mínimos locais.

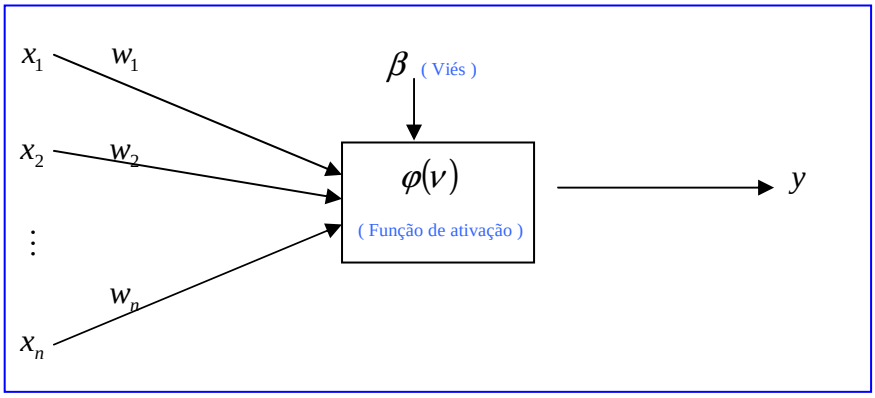
Porém, em 1998, foi desenvolvido uma técnica de treinamento de RNAs a qual utiliza o método de controle por Modos Deslizantes (Parma et al., 1998), o qual controla a convergência da rede para o ponto de mínimos erro e derivada. Porém, alcançar o

ponto de mínimo global implica em se super-ajustar a rede ao conjunto de treinamento o que pode provocar um efeito denominado *overfitting*, o qual impede de a rede a obter respostas coerentes a um conjunto de dados não conhecidos *a priori*. Essa capacidade, a de obter respostas coerentes, é denominada capacidade de generalização.

Segundo Bartlett (1997), uma rede superdimensionada pode vir a apresentar uma boa capacidade de generalização a partir do correto dimensionamento dos seus parâmetros de ajuste ou pesos. Por conta disto vários algoritmos foram desenvolvidos na tentativa de modificar, não a estrutura original da rede, mas sim a amplitude dos parâmetros da mesma, alterando assim a sua capacidade de generalização (Costa, 2003; Costa, 2006).

O método Multi-Objetivo (Teixeira et al., 2000) realiza o equilíbrio entre a norma dos pesos e o erro do treinamento das redes garantindo assim a capacidade de generalização do modelo. Este algoritmo gera um conjunto de soluções com normas variadas e com o menor erro para cada valor de norma, selecionando aquele com a melhor resposta em relação ao conjunto.

Figura 4.1 – Diagrama esquemático de um nodo *Perceptron*



4.2 Treinamento das Redes Neurais Artificiais

A propriedade mais importante de uma rede neural é a sua habilidade de aprender a partir de um conjunto pequeno de dados e de generalizar a informação

prendida. Uma rede neural aprende através de um processo iterativo de ajustes aplicados aos parâmetros da RNA com o objetivo de aproximar uma função pré-estabelecida aos dados de entrada. Este processo iterativo é denominado Treinamento.

Tal processo é normalmente interrompido quando:

- O número máximo de iterações é alcançado;
- O erro de treinamento ou validação torna-se menor do que o *erro tolerável* (previamente definido).

Para a sua realização, divide-se o banco dados em 3 sub-bancos de dados, o primeiro com 60% dos dados do banco original é denominado conjunto de treinamento, o segundo com 20% dos mesmos é denominado conjunto de validação e o terceiro a ser composto pelos dados restantes é denominado conjunto de teste. Os dois primeiros conjuntos visam ajustar os parâmetros da rede, enquanto que o último visa verificar a capacidade de generalização da rede.

Em seguida, é necessário inicializar os pesos da rede. Após a inicialização do vetor dos pesos, *w*, o algoritmo de treinamento, o qual é um algoritmo iterativo, irá calcular o vetor de ajuste dos parâmetros, Δw, o qual será somado ao vetor de pesos atual. A equação do ajuste é:

$$w_{(k+1)} = w_{(k)} + \Delta w \tag{4.1}$$

4.3 Método do Gradiente Descendente

O método do gradiente descendente realiza o ajuste dos pesos no sentido contrário do vetor do gradiente da função de custo, de forma a minimizá-la. Definindo um conjunto de treinamento formado pelos pares $\{(x_i, d_i)\}_{i=1}^{n}$, onde x_i é o i-ésimo vetor de entrada e d_i é a i-ésima saída desejada, e *n* o número de elementos do conjunto de treinamento, a função de custo a ser minimizada é a soma dos quadrados dos erros, descrita pela Equação (4.2).

$$J = \frac{\sum_{i=1}^{n} (y_i - d_i)^2}{2} \tag{4.2}$$

onde,

d_i é a i-ésima saída desejada e

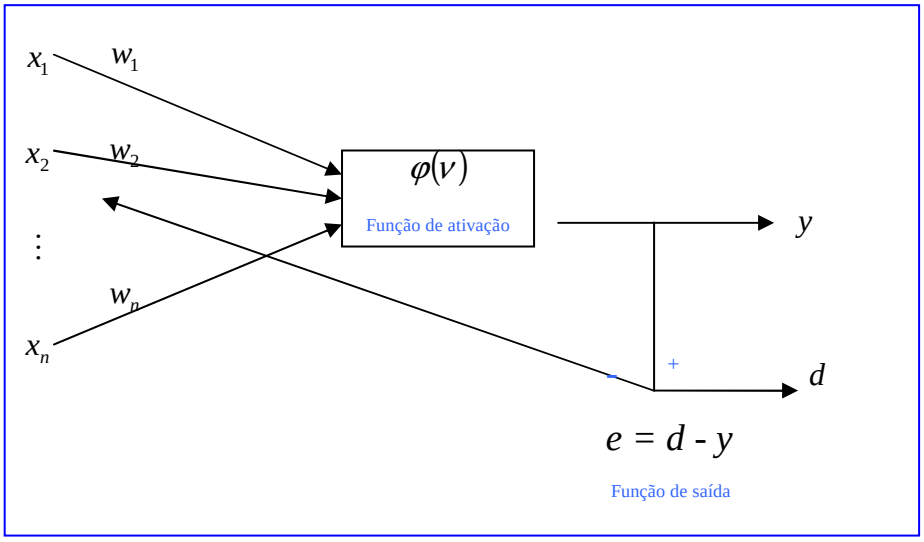
y_i é a i-ésima saída da RNA.

A partir de uma condição inicial qualquer do vetor de pesos, $w_{(0)}$, o ajuste a ser aplicado nos mesmos é definido como sendo $\Delta w \propto -\nabla J$, onde ∇J é o gradiente da função de custo.

Como o ajuste dos pesos depende do cálculo do gradiente, a equação da rede (ou função objetivo) deve ser diferenciável e contínua.

4.3.1 ADALINE

Figura 4.2 – Diagrama esquemático de um nodo Adaline



O modelo Adaline (Widrow e Hoff, 1960), denominado de ADAptive LInear Neuron, surgiu quase que simultaneamente ao *perceptron* (Rosenblatt, 1958). Ambos são baseados na idéia de se ter elementos de processamento executando operações de soma ponderada e depois compará-la com um valor limiar.

O algoritmo descrito por Widrow e Hoff é conhecido como *regra delta* e tem extrema importância na área de RNAs, já que foi ele que deu origem ao algoritmo *backpropagation* para treinamento de *perceptrons* de múltiplas camadas. Este método utiliza o gradiente da função de custo da saída linear do nodo, ou seja, antes da aplicação da função de ativação.

Definindo-se a função de custo através da Equação (4.2) e a constante de proporcionalidade, η, a equação de ajuste dos pesos do nodo adaline é dada por:

$$\Delta w_i = -\eta \nabla J \qquad (4.3)$$

onde,

$\nabla J = \frac{\partial J}{\partial w_i}$.

Utilizando a regra da cadeia temos que:

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial w_i}. \tag{4.4}$$

Desenvolvendo as derivadas temos que:

$$\frac{\partial J}{\partial w_i} = -x_i \left(d - \left(w_0 + w_1 x_1 + \cdots + w_n x_n\right)\right). \tag{4.5}$$

E definindo o erro, $e_{(k)}$, como sendo a diferença entre a saída desejada e a saída do nodo temos a seguinte equação final de ajuste dos pesos:

$$w_{(k+1)} = w_{(k)} + \eta e_{(k)} w_{(k)} \tag{4.6}$$

4.3.2 Regra Delta

Após o surgimento da *Regra de Widrow-Hoff* para o treinamento do nodo Adaline, a aplicação do método do gradiente descendente para o ajuste dos pesos baseado na saída da função de ativação fez surgir um novo conjunto de modelos e por conseguinte, uma nova regra de treinamento conhecida como *regra delta*. Para que a *regra delta* possa ser aplicada à saída de uma função de ativação, esta deve ser diferenciável. A *regra delta* pode então ser representada da seguinte forma: $y = f\left(h\right)$, onde $h = wx^T$.

Seja um nodo com vetor de entrada *x* e vetor de pesos *w*, ambos de p-dimensionais. A saída do mesmo é definida por:

$$y_{(k)} = f\left(h_{(k)}\right),$$

onde,

$h_{(k)} = \sum_{i=1}^{p} \left(w_{i(k)} x_{i(k)}\right)$.

O gradiente da saída em relação aos pesos, segundo a regra da cadeia, é dado por:

$$\frac{\partial y_{(k)}}{\partial w} = \frac{\partial f\left(h_{(k)}\right)}{\partial h_{(k)}} \cdot \frac{\partial h_{(k)}}{\partial w}, \tag{4.7}$$

portanto,

$$\frac{\partial y_{(k)}}{\partial w} = f'\left(h_{(k)}\right) \cdot x_{(k)}.$$

Aplicando o método na função de custo definida pela Equação (4.2), o gradiente do erro, pode ser expresso pela seguinte expressão:

$$\nabla J = \frac{\partial J}{\partial e} \cdot \frac{\partial e}{\partial y_{(k)}} \cdot \frac{\partial y_{(k)}}{\partial w_{(k)}},$$

$$\nabla J = -e \cdot f'\left(h_{(k)}\right) \cdot x_{(k)}$$

onde, $e = d_{(k)} - y_{(k)}$.

Como o ajuste é realizado no sentido contrário à direção do gradiente e, acrescentando a ele a taxa de aprendizado, η, a regra de atualização dos pesos, quando a função de ativação é diferenciável, passa ser expressa por:

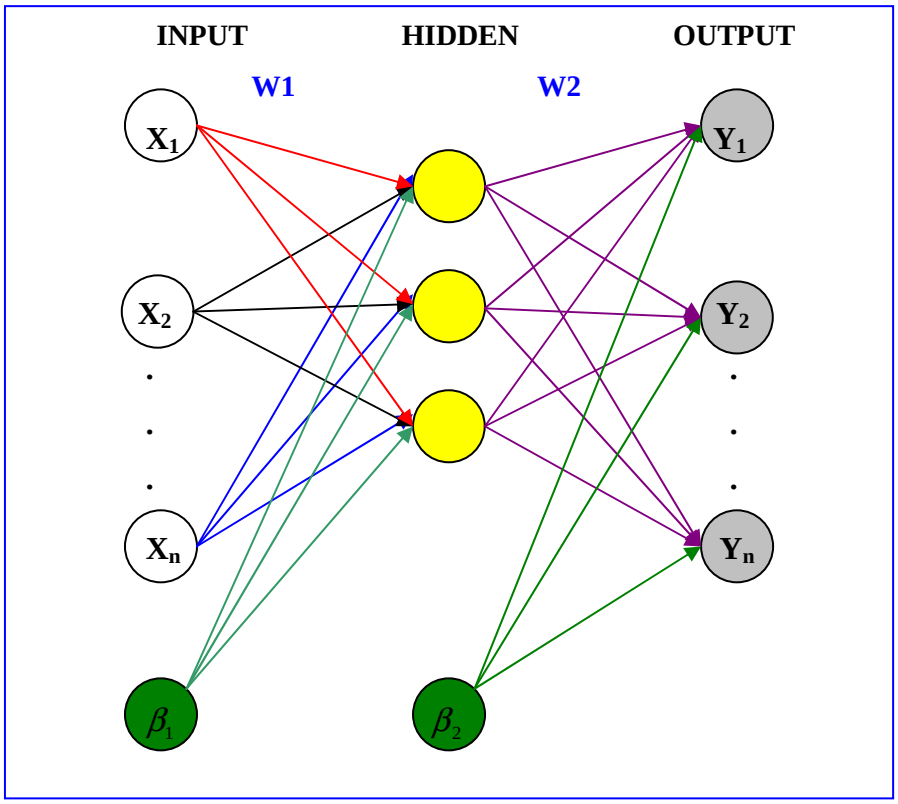
$$w_{(k+1)} = w_{(k)} + \eta e_{(k)} f'\left(h_{(k)}\right) w_{(k)} \tag{4.8}$$

Podemos notar que quando a função de ativação for uma função linear, a sua derivada será uma constante, reduzindo assim a *Regra Delta* à *Regra Widrow-Hoff*.

4.4 Redes Neurais Artificiais do tipo Multi Layer Perceptron

As redes *Multi Layer Perceptron* (MLP) são caracterizadas por uma camada de entrada, uma de saída e 1 ou mais camadas intermediárias ou ocultas, conforme ilustrado na Figura 4.3. Segundo Cybenko (1989) uma rede com uma camada intermediária é capaz de implementar qualquer função contínua, enquanto que a utilização de duas camadas intermediárias permite a aproximação de qualquer função. As redes MLP têm sido aplicadas com sucesso para resolver diversos problemas complexos através de seu treinamento supervisionado com um algoritmo muito popular conhecido como algoritmo *back-propagation*.

Figura 4.3: Rede Neural Artificial do tipo MLP



Neste trabalho, nos restringimos a uma rede MLP com uma camada intermediária, onde todos os nodos de entrada estão ligados a todos os nodos da camada intermediária e estes por sua vez estão todos conectados ao nodo da camada de saída. Nos nodos da camada intermediária uma transformação não-linear é realizada sobre a soma ponderada das entradas. Considere o i-ésimo vetor de entradas $x_i = [x_{i1}, \cdots, x_{iP}]^T$, onde P é o número de covariáveis do modelo. Considere também uma matriz $W1$ de pesos, onde w_{ph} é o peso associado à p-ésima covariável e ao h-ésimo nodo da camada intermediária. Considere também os vetores $W2 = [w_1, \cdots, w_H]$ e $\beta 1 = [\beta_1, \cdots, \beta_H]^T$, os vetores de pesos e constantes associados a cada nodo da camada intermediária respectivamente e H o número de nodos na camada intermediária. Por fim, $\beta 2$ uma constante. A entrada do h-ésimo nodo da camada intermediária é dada pela projeção linear $W2^T x_i$ enquanto que a sua saída é $f(W2^T x_i)$, onde $f(.)$ é a 'função de ativação'. A função de ativação mais utilizada é a função logística $f(z) = 1/(1+e^{-z})$. Existem outras funções de ativação como, por exemplo, a tangente hiperbólica, a qual será utilizada neste trabalho. A saída da rede é uma soma ponderada (por $W2$) da saída dos

nodos da camada intermediária mais a constante $\beta 2$, já que nossa rede possui saída linear. Portanto podemos representar uma rede neural com função de ativação sigmoidal (*tanh*) e saída linear como:

$$y_i = g(x_i;\theta) = \sum_{h=1}^{H} w2_h \tanh\left(\sum_{p=1}^{P} w1_{hp} x_{ip} + \beta 1_h\right) + \beta 2 \qquad (4.9)$$

onde, θ é o vetor de parâmetros da rede $[\mathrm{W1}, \mathrm{W2}, \beta 1, \beta 2]^{\mathrm{T}}$

O número total de parâmetros em uma rede é $m = H(P+2)+1$. Podemos definir $y = g(x,\theta) = [g(x_1,\theta),\cdots,g(x_n,\theta)]^T$ como sendo o vetor composto por todas as *n* saídas da rede.

Os principais algoritmos utilizados para o treinamento de redes neurais do tipo MLP são baseados no método do gradiente descendente (Haykin, 2001). Uma vez que as funções de ativação para cada camada são contínuas e diferenciáveis, a saída da rede pode ser expressa através de uma equação, ou seja, é possível calcular o gradiente da função custo em relação aos parâmetros de ajuste da rede.

4.4.1 Algoritmo *Backpropagation*

Existem diversos algoritmos para o treinamento de redes do tipo MLP. Dentre estes o mais conhecido é o *backpropagation* (Rumelhart, Hilton e Williams, 1986). Este algoritmo é baseado na regra delta, sendo por isto também denominado regra delta generalizada. O algoritmo *backpropagation* propõe uma forma de definir o erro dos nodos das camadas intermediárias possibilitando o ajuste de seus pesos, através do método do gradiente descendente.

A função de custo a ser minimizada também é a definida na Equação (4.2). Considerando η, como sendo a taxa de aprendizado e sendo a saída da rede *y* dada pela Equação (4.9), temos então que a equação geral de ajuste dos pesos (parâmetros) da rede é dada por:

$$w_{(k+1)} = w_{(k)} - \eta \nabla J_{(k)}, \qquad (4.10)$$

onde, o gradiente da função de custo, ∇J, utilizando a regra da cadeia, é dado por:

$$\nabla J_{(k)} = \frac{\partial J}{\partial w_{(k)}} = \frac{\partial J}{\partial y_{(k)}} \cdot \frac{\partial y_{(k)}}{\partial w_{(k)}}. \qquad (4.11)$$

Logo, quatro equações de ajuste dos pesos, podem ser definidas para cada vetor de pesos associados às camadas escondida e de saída da rede como sendo:

$$
\begin{aligned}
w1_{hp(k+1)} &= w1_{hp(k+1)} - \eta \frac{\partial J_{(k)}}{w1_{hp(k+1)}} \\
w2_{h(k+1)} &= w2_{h(k)} - \eta \frac{\partial J_{(k)}}{w2_{h(k)}} \\
\beta 1_{h(k+1)} &= \beta 1_{h(k)} - \eta \frac{\partial J_{(k)}}{\beta 1_{h(k)}} \\
\beta 2_{(k+1)} &= \beta 2_{(k)} - \eta \frac{\partial J_{(k)}}{\beta 2_{(k)}}
\end{aligned}
\tag{4.12}
$$

onde,

$$
\begin{aligned}
\frac{\partial J_{(k)}}{w1_{hp(k)}} &= \frac{\partial J_{(k)}}{\partial y_{(k)}} \cdot \frac{\partial y_{(k)}}{w1_{hp(k)}} \\
\frac{\partial J_{(k)}}{w2_{h(k)}} &= \frac{\partial J_{(k)}}{\partial y_{(k)}} \cdot \frac{\partial y_{(k)}}{w2_{h(k)}} \\
\frac{\partial J_{(k)}}{\beta 1_{h(k)}} &= \frac{\partial J_{(k)}}{\partial y_{(k)}} \cdot \frac{\partial y_{(k)}}{\beta 1_{h(k)}} \\
\frac{\partial J_{(k)}}{\beta 2_{(k)}} &= \frac{\partial J_{(k)}}{\partial y_{(k)}} \cdot \frac{\partial y_{(k)}}{\beta 2_{(k)}}
\end{aligned}
\tag{4.13}
$$

onde,

$$
\begin{aligned}
\frac{\partial J_{(k)}}{\partial y_{(k)}} &= d_{(k)} - y_{(k)} = e_{(k)} \\
\frac{\partial y_{(k)}}{w1_{hp(k)}} &= w2_{h(k)} \cdot \sec h\left(w1_{hp(k)} x_p + \beta 1_{h(k)}\right) \cdot x_p \\
\frac{\partial y_{(k)}}{w2_{h(k)}} &= \tanh\left(\sum_{p=1}^{P} w1_{hp(k)} x_p + \beta 1_{h(k)}\right) \\
\frac{\partial y_{(k)}}{\beta 1_{h(k)}} &= w2_{h(k)} . \sec h\left(\sum_{p=1}^{P} \left(w1_{hp(k)} x_p + \beta 1_{h(k)}\right)\right) \\
\frac{\partial y_{(k)}}{\beta 2_{(k)}} &= 1
\end{aligned}
\quad .
\tag{4.14}
$$

Denotando de forma matricial temos:

$$\begin{aligned}
&\frac{\partial J_{(k)}}{W1_{(k)}} = -\left(W2_{(k)}^{T} \cdot e\right) \otimes \left(\sec h\left(W1 \cdot X + \beta 1_{(k)} \cdot 1_n\right)\right) \cdot X^{T} \\
&\frac{\partial J_{(k)}}{W2_{(k)}} = -e \cdot \left(\tanh\left(W1 \cdot X + \beta 1_{(k)} 1_n\right)\right)^{T} \\
&\frac{\partial J_{(k)}}{\beta 1_{(k)}} = -\left(W2_{(k)}^{T} \cdot e\right) \otimes \left(\sec h\left(W1 \cdot X + \beta 1_{(k)} \cdot 1_n\right)\right) \\
&\frac{\partial J_{(k)}}{\beta 2_{(k)}} = -e
\end{aligned} \quad . \qquad (4.15)$$

onde, $e_{(k)} = \left[\left(d_1 - y_{1(k)}\right), \cdots, \left(d_n - y_{n(k)}\right)\right]$, $1_n = [1, \cdots, 1]$ de dimensão n, $X = [x_1, \cdots, x_n]$ é uma matriz $p \times n$, tal que p é o número de covariáveis e n o número de dados da amostra e '$\otimes$' simboliza o produto matricial termo a termo (se $A \otimes B = C$, então $c_{ij} = a_{ij} \times b_{ij}$).

As desvantagens do algoritmo *backpropagation* consistem em sua baixa velocidade de convergência e na sua limitação ao se deparar com mínimos locais já que se trata de um algoritmo o qual depende somente do gradiente local, ou seja, caso haja um ponto de mínimo local nas proximidades ele ficará preso.

Uma forma de se melhorar a resposta da rede utilizando o *backpropagation* é treinando diversas redes com pesos inicializados aleatoriamente o que leva a rede a uma maior chance de obter uma rede cuja solução seja próxima do mínimo global, porém isto implica em um grande custo computacional e não é possível garantir uma boa solução.

Vários métodos foram desenvolvidos a partir deste algoritmo visando evitar a convergência para regiões de mínimo local. O *RProp* (Riedmiller e Braun, 1993) utiliza o sinal do gradiente e não o seu valor para realizar a correção dos pesos. Já o *QuickProp* (Fahlman,1988) aproxima a superfície do erro por uma parábola em função dos pesos. Tal ajuste é realizado de forma que o erro mínimo da parábola seja alcançado. Porém nem sempre esta superfície pode ser modelada por uma parábola. Os métodos de *Taxa Adaptativa* (Silva e Almeida,1990 e Tollenaere, 1990) utilizam técnicas para o ajuste da taxa de aprendizado. Porém, nenhum destes métodos é capaz de garantir uma convergência ao mínimo global.

4.4.2 Algoritmo Levenberg-Marquardt

O algoritmo *Levenberg-Marquardt* (Marquardt,1963) é um exemplo de algoritmo que visa minimizar a função custo, buscando o mínimo global, utilizando técnicas de otimização mais complexas. Algoritmos como o *Levenberg-Marquardt* apresentam um custo computacional mais elevado, porém apresentam um ganho significativo em termos do número de iterações. Por envolver cálculos mais complexos, um aumento no número de parâmetros da rede ou do tamanho da amostra provoca um aumento no tempo de convergência o que pode vir a causar a perda da eficiência do modelo.

Utilizando uma aproximação do Método de Newton, a função de custo é definida em função de um vetor de parâmetros *x*:

$$V_{(x)} = \sum_{i=1}^{N} e_i^2(x), \tag{4.16}$$

onde, *x* é um vetor composto pelos parâmetros da rede arranjados vetorialmente.

O ajuste do vetor de parâmetros utilizando o Método de Newton é descrito por:

$$x_{(k+1)} = x_{(k)} - \lambda H_{(x_{(k)})}^{-1} \nabla f\left(x_{(k)}\right), \tag{4.17}$$

onde, $H_{(x_{(k)})}$ é a matriz Hessiana, $\nabla f\left(x_{(k)}\right)$ é o vetor gradiente no ponto $x_{(k)}$ e λ define o passo do algoritmo.

Um ajuste semelhante para o vetor de pesos é definido por:

$$\Delta x = -\left[\nabla^2 V_{(x)}\right]^{-1} \nabla V_{(x)}, \tag{4.18}$$

onde, $\nabla^2 V_{(x)}$ é a matriz Hessiana de $V_{(x)}$ e $\nabla V_{(x)}$ é o gradiente da função $V_{(x)}$.

Utilizando o Método de Gauss-Newton, obtêm-se as seguintes aproximações:

$$\begin{aligned} \nabla V_{(x)} &= J_{(x)}^T \cdot e_{(x)} \\ \nabla^2 V_{(x)} &= J_{(x)}^T \cdot J_{(x)} \end{aligned} \tag{4.19}$$

onde, $J_{(x)}$ é a matriz jacobinana.

O ajuste de *x*, neste caso, é obtido através de uma modificação do Método de Newton:

$$\Delta x = -\left[J_{(x)}^T \cdot J_{(x)} + \mu \cdot I\right]^{-1} J_{(x)}^T \cdot e_{(x)}, \tag{4.20}$$

Como podemos observar pela Equação (4.20) para valores grandes de μ, o método descrito é do gradiente descendente, caso contrário, o de Gauss-Newton. O algoritmo segue um padrão para o ajuste do parâmetro μ: ele é multiplicado por um fator α quando o ajuste provoca um aumento na função de erro e dividido no caso de uma diminuição.

4.4.3 Algoritmo Modos Deslizantes

O algoritmo proposto por Parma, Menezes e Braga (1999), é baseado na teoria do controle por modos deslizantes, SMC – *Sliding Mode Control* (Utkin, 1978), o qual agrega velocidade de treinamento e robustez. Mesmo sendo um algoritmo que utiliza o gradiente da função custo no ponto, ele apresenta um melhor desempenho do que o *backpropagation.* Outra vantagem é a teoria envolvida na obtenção analítica dos parâmetros de treinamento, com a qual é possível definir limites para os parâmetros de aprendizado da rede de forma a garantir a convergência da rede. Mas a grande vantagem do *Modos Deslizantes* consiste na sua adaptabilidade para qualquer configuração de rede do tipo MLP.

O algoritmo de *Modos Deslizantes*, partindo-se de um estado inicial, controla o estado da rede de forma que as superfícies de deslizamento sejam alcançadas e/ou atravessadas. A partir disso ocorre o que é denominado regime de *Modos Deslizantes* no qual, pelas condições de convergência, é garantida a convergência do estado da rede à origem do espaço de estados, representado pelo erro e derivadas nulas.

Os chamados algoritmos híbridos, como o SMC, permitem incorporar ao treinamento de redes as características de convergência e comportamento de outros métodos. Além disso, esta técnica possibilita além da diminuição do tempo de convergência, uma formulação matemática capaz de garantir a minimização da função custo.

4.4.4 Algoritmo Multi-Objetivo

A maioria dos algoritmos para treinamento de redes, possui como função custo a função descrita pela Equação (4.2), porém, o erro *nulo* alcançado no treinamento representa um modelamento do ruído por parte da rede, outro fator limitante é a

dimensão da rede que pode impedir a convergência a um resultado satisfatório, ou seja, o treinamento pode resultar em um superajuste ou um subajuste. Na tentativa de se solucionar tais problemas, surgem os algoritmos de *pruning* os quais visam simplificar a rede e melhorar a sua resposta. Sob este aspecto, é evidente que existe um equilíbrio entre a dimensão da rede e a resposta desejada.

O algoritmo Multi-Objetivo (Teixeira et al., 2000) apresenta o treinamento de redes do tipo MLP sob uma nova perspectiva: o plano bi-dimensional definido pelo erro e norma e o conjunto Pareto-ótimo. O primeiro passo do algoritmo Multi-Objetivo (MOBJ) é o levantamento do conjunto de soluções ótimas que se localizam no limiar entre as factíveis e as não existentes definidas no espaço de estados representado pelas funções do erro e da norma. A partir deste levantamento, um decisor escolhe a solução de melhor resposta, ou seja, menor erro de validação.

O algoritmo é definido pela seguinte formulação:

- $f_1^* \in \Re^2$: vetor que representa a solução de *underfitting:* norma baixa e erro elevado;
- $f_2^* \in \Re^2$: vetor que representa a solução de *overfitting:* erro baixo e norma elevada;
- $f^* \in \Re^2$: vetor que representa a solução *utópica:* contém a norma de f_1^* e o erro de f_2^*;
- v_k :vetor construído pela combinação de f_1^* e f_2^*.

$$v_k = f^* + \gamma_k \left(f_1^* - f^*\right) + \left(1-\gamma_k\right)\left(f_2^* - f^*\right), \tag{4.21}$$

onde, $0 < \gamma_k < 1$ e *k* é o número de pontos a serem levantados para formar o conjunto Pareto-ótimo. O problema multi-objetivo é então redefinido como um problema mono-objetivo da seguinte forma:

$$w^* = \arg_w \min_{w,\eta} \eta, \tag{4.22}$$

sujeito a:

$$f_i(w) \leq f^* + \eta v_k, \tag{4.23}$$

com $f_1(w)$ definida pela Equação (4.2), $f_2(w)$ definido como a norma dos pesos $f_2(w) = \frac{1}{2}\|w\|^2$ e η uma variável auxiliar.

Reescrevendo o problema mono-objetivo em função das restrições, temos:

$$w^* = \arg_w \min_{w,\eta} \eta ,$$

sujeito a:

$$\begin{cases} g_1(w,\eta) = \frac{1}{2}\sum_{i=1}^{n}(d_i - y_i) - f_1^* - \eta v_k \le 0 \\ g_2(w,\eta) = \frac{1}{2}\|w\|^2 - f_2^* - \eta v_k \le 0 \end{cases} . \qquad (4.24)$$

O algoritmo *Multi-Objetivo* (Teixeira et al., 2000), utiliza o método de otimização Elipsoidal (Shor, 1977) para encontrar a solução do problema mono-objetivo. Para a solução do problema multi-objetivo, ele utiliza um método de otimização denominado *Algoritmo de Relaxação* (Takahashi et al, 1997), o qual apresenta um custo elevado, pois são realizadas *k*-operações de otimização, uma para cada ponto do Pareto-ótimo.

Um segundo método multi-objetivo foi proposto por Teixeira, denominado *Método ε-restrito* (Teixeira, 2001), este método permite expressar o problema multi-objetivo na forma de mono-objetivo, sendo as soluções encontradas via algoritmo elipsoidal. Nesse caso, o objetivo é minimizar a função do erro (função custo), e a norma passando a ser definida como sendo uma restrição.

$$\min_{w \in W} \frac{1}{N_T} \sum_{i=1}^{N_T} (d_i - f(x_i, w))^2 , \qquad (4.25)$$

sujeito a:

$$\|w\| \le \varepsilon., \qquad (4.26)$$

Este método, *Método ε-restrito*, além de tornar desnecessário o cálculo de uma solução via *backpropagatin*, em algumas situações, apresenta uma redução no tempo de cálculo dos pontos pertencentes ao conjunto Pareto.

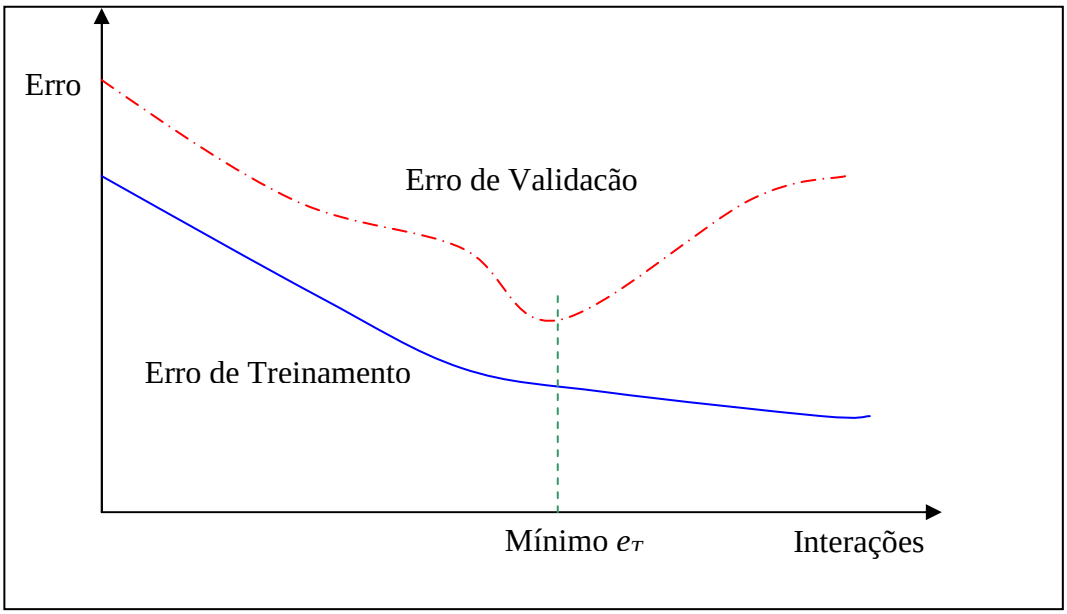
4.5 Interrupção do Treinamento

Interromper o treinamento de uma rede antes de seu término é uma estratégia, denominada *Early Stopping* (Weigend et al., 1990), a qual é utilizada para a obtenção de redes com boa capacidade de generalização, sendo sugerida através do comportamento do erro de generalização em relação a um dado conjunto de validação, C_V , o qual passa

por um ponto de mínimo durante o processo de minimização do erro de treinamento, e_T, em relação ao conjunto de treinamento, C_T.

No processo de treinamento de redes MLP onde os dados apresentam ruídos, pode-se notar que, à medida que o treinamento evolui, o erro de validação, e_V, decresce até um ponto de mínimo e, a partir deste ponto, começa a crescer novamente; apesar de e_T continuar decrescendo, como ilustrado pela Figura 4.4.

Figura 4.4: Dilema entre a Capacidade de Generalização da Rede e a Minimização do Erro



Considerando o comportamento do erro de validação, nota-se que o seu ponto de mínimo não corresponde ao da curva para o erro de treinamento. Conseqüentemente, se a opção for treinar uma rede para que o erro de treinamento seja minimizado, esta rede não apresentará o melhor desempenho possível para os padrões desconhecidos. Ou seja, ao trabalharmos com dados ruidosos, o treinamento excessivo causa o superajuste da rede (*overfitting*). Para se evitar este efeito, deve-se treinar a rede até que o e_V atinja o seu mínimo. Neste ponto, o e_T não é mínimo, mas é o menor possível para os padrões de validação, logo o treinamento deve ser interrompido.

Observando a Figura 4.4, pode-se notar que o processo de treinamento pode ser dividido em duas partes. A primeira vai do início do treinamento até o ponto de mínimo da curva de erro de validação, nesta fase a rede se adapta somente às principais características dos dados, ou seja, aprende a função geradora. Na fase seguinte, na

medida em que o treinamento prossegue o ruído também começa a ser mapeado pela mesma.

Caso uma rede tenha complexidade adequada para que apenas a função geradora de C_T seja aprendida, então não há o que se preocupar. Porém, não é isto o que se vê na prática. Na maioria das vezes temos redes mais complexas do que o necessário. Neste caso devemos interromper o treinamento quando o erro de validação atingir o seu mínimo de forma a evitar que o ruído presente nos dados seja modelado. Se uma análise relativa ao valor da norma do vetor de pesos for feita, verifica-se que no ponto de máxima capacidade de generalização o vetor de pesos terá uma norma menor que no ponto onde o treinamento é mínimo, ou seja, as soluções onde o ruído é modelado possuem normas mais elevadas.

Um fato importante no treinamento de uma RNA com função de custo dada pela Equação (4.2) é que à medida que o treinamento prossegue, a magnitude dos pesos também aumenta. Ou seja, quando interrompemos o treinamento, o que estamos fazendo é impor uma restrição para a norma do vetor de pesos, evitando que ela cresça em demasia, o que faria com que a rede modelasse não só a função geradora dos dados, mas também o ruído. Porém, para que o treinamento possa ser interrompido faz-se necessária uma avaliação do erro de validação. Tal avaliação pode ser feita dividindo o banco de dados em duas partes. A primeira é utilizada para o treinamento e a segunda para o cálculo do erro de validação a cada iteração do treinamento. Nem sempre o erro de validação tem comportamento monotonicamente decrescente até o seu ponto de mínimo, podendo existir mínimos locais, o que dificulta a determinação do melhor momento de interrupção.

Uma outra estratégia de escolha de um modelo com boa capacidade de generalização é conhecida como *Validação Cruzada*, sobre a qual descreveremos a seguir.

4.6 Validação Cruzada (*Cross-Validation*)

Uma outra maneira de se avaliar o erro de generalização para se conseguir redes com boa capacidade de generalização é conhecida como Validação Cruzada (Stone, 1978). Esta estratégia é apropriada para melhorar a capacidade de generalização de redes com complexidade acima da necessária, evitando desta forma o *overfitting.* A

Validação Cruzada é uma melhoria da validação por meio da divisão da amostra, como relatada na seção anterior, permitindo que todo o conjunto de dados seja usado para treinamento. Este método é melhor do que o *Early Stopping,* principalmente, nos casos onde o conjunto de dados é pequeno (Goutte, 1997). A desvantagem da Validação Cruzada é a necessidade de se treinar várias redes. Existem duas formas de se implementar a Validação Cruzada.

Na primeira delas, denominada *k-fold Cross-Validation*, o conjunto de dados é dividido em *k* partes de forma a constituir *k* conjuntos distintos de treinamento e validação, os quais são utilizados para o treinamento de *k* redes. Se $k = n$, onde *n* é o número total de indivíduos do banco de dados, então este método é denominado *leave-one-out Cross-Validation*.

Leave-one-out Cross-Validation apresenta boas estimativas para o erro de generalização para funções de custo contínuas. No caso de funções de custo descontínuas este método pode apresentar baixo desempenho. Para este último caso, *k-fold Cross-Validation* pode apresentar melhores resultados.

Sendo n o tamanho total do conjunto de dados, cada conjunto de treinamento é constituído de $\frac{n}{k}(k-1)$ indivíduos e cada conjunto de validação e cada conjunto de validação é constituído por $\frac{n}{k}$ indivíduos, os quais são utilizados para o cálculo do erro de generalização.

A Validação Cruzada tem a vantagem, em relação ao *Early Stopping,* de trabalhar com amostras distintas sobre o mesmo conjunto de dados. Como desvantagens podemos citar a necessidade de se constituir diversos conjuntos de dados, de se treinar várias redes as quais estão sujeitas aos parâmetros de treinamento e ainda poderem ficar estagnadas em mínimos locais. Esta técnica fornece melhores resultados quando o conjunto de dados não é muito pequeno.

4.7 Conclusões do Capítulo

Este capítulo teve como objetivo introduzir o leitor à teoria de redes neurais, comentando um pouco sobre o que são, como surgiu e abordando diversos algoritmos os quais tem por objetivo obter modelos de RNA com uma boa capacidade generalização. Foram abordadas também questões importantes ao processo de

treinamento das redes neurais como a complexidade, a capacidade de generalização e o superajuste das mesmas (*overfitting*)

Para o caso do *Early Stopping*, tem-se o problema da amostragem única para o conjunto de validação e de treinamento, além do erro de generalização que pode possuir mínimos locais. Já a *Validação Cruzada*, apresenta melhores resultados quando se trabalha com banco de dados maiores. Outro ponto importante ao se trabalhar com divisões de banco de dados é a determinação do tamanho de cada uma destas partes e de como elas foram amostradas.

O algoritmo MOBJ não altera a estrutura da rede ao longo do treinamento. Ele, através da imposição de limites para os parâmetros da RNA, é capaz de encontrar uma boa solução para a mesma, que a complexidade efetiva da rede seja muito elevada.

Este capítulo fecha a parte de revisão bibliográfica deste estudo. No próximo capítulo apresentaremos o modelo de *Redes Neurais Generalizadas*. Este modelo visa unir as teorias de Modelagem de dados via *Modelos de Regressão* com a modelagem computacional não-paramétrica das *Redes Neurais Artificiais.*

Capítulo 5

Redes Neurais Generalizadas

Conforme descrito, a solução de problemas via RNAs é bem atrativa já que a mesma consegue extrair dos dados, automaticamente, características sobre quais co-variáveis são mais importantes e suas interações sem que para isso seja necessário inferir previamente sobre essas relações. A idéia dos modelos de Redes Neurais Generalizadas (RNGs) é agregar a informação da função de verossimilhança à modelagem não-paramétrica computacional das RNAs.

Seja a log-verossimilhança de um Modelo Linear Generalizado:

$$l(d,\theta) = \sum_{i=1}^{n} \frac{[d_i\theta_i - b(\theta_i)]}{a(\phi)} + c(d_i,\phi) \qquad (5.1)$$

onde θ é o parâmetro referente à média, $\mu = b'(\theta)$, e ϕ é o parâmetro referente à variabilidade. Uma RNG é obtida a partir da associação do parâmetro canônico, θ, à saída de uma rede neural com função de ativação sigmoidal e saída linear, podendo ser expressa na forma:

$$\theta_i = \sum_{h=1}^{H} w2_h.\tanh\left(\sum_{p=1}^{P} w1_{hp}\, x_p + \beta 1_h\right) + \beta 2 \qquad (5.2)$$

ou na forma matricial,

$$\theta = W2.\tanh(W1.X + \beta 1 \cdot 1_n) + \beta 2 \qquad (5.3)$$

Neste caso, pode-se utilizar o algoritmo de treinamento *back-propagation* para realizar a atualização dos pesos via método do gradiente descendente. Este método terá como objetivo, maximizar a função de verossimilhança ou minimizar a função desvio com base em um conjunto de treinamento $T = \{(x_i, d_i)\}_{i=1}^{n}$. Dada uma certa condição

inicial, $w_{(0)}$, para os pesos da rede, deseja-se obter a direção do ajuste a ser aplicado no vetor de pesos de forma a encontrarmos a direção para a solução, a qual maximize a verossimilhança. A direção do ajuste no passo *k* pode ser obtida pelo gradiente da função de custo no ponto $w_{(k)}$. A fim de maximizar o logaritmo da verossimilhança, o ajuste neste caso é realizado na mesma direção do gradiente no ponto $w_{(k)}$, ou seja, $w_{(k+1)} = w_{(k)} + \Delta w_{(k)}$ onde: $\Delta w_{(k)} = \nabla l(d, \theta)$.

As equações de ajuste do vetor de parâmetros via método do gradiente considerando um modelo neural como saída do parâmetro canônico, podem ser obtidas na forma:

$$\begin{aligned} w1_{hp(k+1)} &= w1_{hp(k)} - \eta r \frac{\partial l\left(d, \theta_{(k)}\right)}{w1_{hp(k)}} \\ w2_{h(k+1)} &= w2_{h(k)} - \eta r \frac{\partial l\left(d, \theta_{(k)}\right)}{w2_{h(k)}} \\ \beta 1_{h(k+1)} &= \beta 1_{h(k)} - \eta r \frac{\partial l\left(d, \theta_{(k)}\right)}{\beta 1_{h(k)}} \\ \beta 2_{(k+1)} &= \beta 2_{(k)} - \eta r \frac{\partial l\left(d, \theta_{(k)}\right)}{\beta 2_{(k)}} \end{aligned} \tag{5.4}$$

onde, *r* é uma variável aleatória uniformemente distribuída no intervalo $[0,1]$, a qual incorpora um comportamento aleatório à taxa de aprendizado.

As derivadas da log-verossimilhança em relação aos parâmetros são definidas da seguinte forma:

$$\begin{aligned} \frac{\partial l\left(d, \theta_{(k)}\right)}{w1_{hp(k)}} &= \frac{\partial l\left(d, \theta_{(k)}\right)}{\partial \theta_{(k)}} \cdot \frac{\partial \theta_{(k)}}{w1_{hp(k)}} \\ \frac{\partial l\left(d, \theta_{(k)}\right)}{w2_{h(k)}} &= \frac{\partial l\left(d, \theta_{(k)}\right)}{\partial \theta_{(k)}} \cdot \frac{\partial \theta_{(k)}}{w2_{h(k)}} \\ \frac{\partial l\left(d, \theta_{(k)}\right)}{\beta 1_{h(k)}} &= \frac{\partial l\left(d, \theta_{(k)}\right)}{\partial \theta_{(k)}} \cdot \frac{\partial \theta_{(k)}}{\beta 1_{h(k)}} \\ \frac{\partial l\left(d, \theta_{(k)}\right)}{\beta 2_{(k)}} &= \frac{\partial l\left(d, \theta_{(k)}\right)}{\partial \theta_{(k)}} \cdot \frac{\partial \theta_{(k)}}{\beta 2_{(k)}} \end{aligned} \tag{5.5}$$

onde, $\frac{\partial l(d,\theta)}{\partial \theta}$ é a derivada da função custo em relação ao parâmetro canônico, θ, que neste caso é a função de log-verossimilhança, em relação à rede e informa a contribuição da verossimilhança para o modelo de RNGs. A derivada $\frac{\partial l(d,\theta)}{\partial \theta}$ possui uma forma geral conhecida para os MLGs com ligação canônica dada por: $\frac{\partial l(d,\theta)}{\partial \theta} = d - \mu$, onde $d = [d_1, \cdots, d_n]^T$ com d_i sendo o valor desejado para i-ésima saída da rede e $\mu = [\mu_1, \cdots, \mu_n]^T$ com $\mu_i = b'(\theta_i)$ sendo o termo definido unicamente pela forma paramétrica associada à variável resposta.

O segundo termo das equações de ajuste dos pesos representa a derivada parcial da saída linear da rede MLP em relação aos vetores de pesos (ou parâmetros) definidas pelas Equações (4.14) e expressas em sua forma matricial pelas Equações (4.15).

Portanto, a forma do ajuste dos pesos para uma RNG é definida pelas seguintes expressões:

$$
\begin{aligned}
w1_{hp(k)} &= \left(d_{(k)} - y_{(k)}\right) \cdot w2_{h(k)} \cdot \sec h^2\left(w1_{hp(k)} x_p + \beta 1_{h(k)}\right) \cdot x_p \\
\beta 1_{h(k)} &= \left(d_{(k)} - y_{(k)}\right) \cdot w2_{h(k)} . \sec h^2\left(\sum_{p=1}^{P}\left(w1_{hp(k)} x_p + \beta 1_{h(k)}\right)\right) \\
w2_{h(k)} &= \left(d_{(k)} - y_{(k)}\right) \cdot \tanh\left(\sum_{p=1}^{P} w1_{hp(k)} x_p + \beta 1_{h(k)}\right) \\
\beta 2_{(k)} &= \left(d_{(k)} - y_{(k)}\right)
\end{aligned}
\tag{5.6}
$$

onde,

sech^2 é a derivada da tangente hiperbólica e

$d_{(k)} - y_{(k)}$ é a medida do erro entre a resposta desejada e a resposta predita pelo modelo.

Algoritmo de uma RNG

1. Inicialize os pesos: $W1$, $W2$, $\beta 1$, $\beta 2$.

2. Defina a taxa de aprendizado η.

3. Divida o Banco de dados em 3 sub-bancos: Treinamento, Validação e Teste.

4. Para i de 1 até M calcule

(a) Calcule os gradientes utilizando o conjunto de treinamento

(b) $\eta_0 = \eta * r$, onde r ~ U(0,1)

(c) Faça o ajuste dos pesos

$$W1_{(i+1)} = W1_{(i)} - \eta_0 \nabla W1_{(i)}$$
$$W2_{(i+1)} = W2_{(i)} - \eta_0 \nabla W2_{(i)}$$
$$\beta 1_{(i+1)} = \beta 1_{(i)} - \eta_0 \nabla \beta 1_{(i)}$$
$$\beta 2_{(i+1)} = \beta 2_{(i)} - \eta_0 \nabla \beta 2_{(i)}$$

(d) Calcule e armazene a função Desvio para os conjuntos de Treinameto e Validação

(e) Se desvio para o conjunto de treinamento for menor do que aquele calculado no passo anterior, mantenha a atualização dos pesos, caso contrário desfaça-a.

(f) Armazene os pesos com menor Desvio de Validação

5. Para o conjunto de Teste calcule o Desvio com os pesos armazenados em 4(f)

5.1 O Ajuste de uma RNG para o Modelo de Regressão de Cox

O modelo de Redes Neurais Generalizadas pode também ser ajustado aos dados para a análise de sobrevivência, os quais, na maioria das vezes, são ajustados via modelos de regressão de Cox. Este ajuste é realizado fazendo a função de verossimilhança parcial dada pela Equação (3.5) como sendo a função custo.

Seja a função de verossimilhança parcial como sendo:

$$L(\beta) = \prod_{i=1}^{n} \left(\frac{\exp\left(x_i^T \beta\right)}{\sum_{l \in C_i} \exp\left(x_l^T \beta\right)} \right)^{\delta_i} \qquad (5.7)$$

Fazendo $\theta_i = x_i^T \beta$, a função de log-verossimilhança parcial, ou seja, $\log(L(\beta))$, é dada por:

$$l(\theta) = \sum_{i=1}^{n} \delta_i \left(\theta_i - \log\left(\sum_{l \in C_i} \exp(\theta_l) \right) \right) \tag{5.8}$$

Utilizando o método do gradiente descendente as equações de ajuste dos pesos são calculadas de forma análoga às apresentadas pelas Equações (5.4) e (5.5). Desta forma as equações de ajuste dos pesos serão definidas pelas seguintes expressões:

$$\begin{aligned}
w1_{hp(k+1)} &= w1_{hp(k)} - \eta r \frac{\partial l(\theta_{(k)})}{w1_{hp(k)}} \\
\beta 1_{h(k+1)} &= \beta 1_{h(k)} - \eta r \frac{\partial l(\theta_{(k)})}{\beta 1_{h(k)}} \\
w2_{h(k+1)} &= w2_{h(k)} - \eta r \frac{\partial l(\theta_{(k)})}{w2_{h(k)}} \\
\beta 2_{(k+1)} &= \beta 2_{(k)} - \eta r \frac{\partial l(\theta_{(k)})}{\beta 2_{(k)}}
\end{aligned} \tag{5.9}$$

Enquanto que a derivada de $l(\theta)$ em relação aos parâmetros é dada por:

$$\begin{aligned}
\frac{\partial l(\theta_{(k)})}{w1_{hp(k)}} &= \frac{\partial l(\theta_{(k)})}{\partial \theta_{(k)}} \cdot \frac{\partial \theta_{(k)}}{w1_{hp(k)}} \\
\frac{\partial l(\theta_{(k)})}{w2_{h(k)}} &= \frac{\partial l(\theta_{(k)})}{\partial \theta_{(k)}} \cdot \frac{\partial \theta_{(k)}}{w2_{h(k)}} \\
\frac{\partial l(\theta_{(k)})}{\beta 1_{h(k)}} &= \frac{\partial l(\theta_{(k)})}{\partial \theta_{(k)}} \cdot \frac{\partial \theta_{(k)}}{\beta 1_{h(k)}} \\
\frac{\partial l(\theta_{(k)})}{\beta 2_{(k)}} &= \frac{\partial l(\theta_{(k)})}{\partial \theta_{(k)}} \cdot \frac{\partial \theta_{(k)}}{\beta 2_{(k)}}
\end{aligned} \tag{5.10}$$

onde, $\frac{\partial l(\theta)}{\partial \theta}$ é a derivada da função custo, que neste caso é o logaritmo da função de verossimilhança parcial, em relação à rede e informa a contribuição da verossimilhança parcial para o modelo de RNGs.

Já o segundo termo das equações de ajuste dos pesos representa a derivada parcial da saída linear da rede MLP em relação aos vetores de pesos.

Estas derivadas não são triviais de serem calculadas como as apresentadas pelas Equações (5.5). Para se ter uma idéia de sua complexidade, apresentaremos a seguir os cálculos de $\frac{\partial l(\theta)}{\partial w1_{hp}}$, $\frac{\partial l(\theta)}{\partial \beta 1_h}$, $\frac{\partial l(\theta)}{\partial w2_h}$, $\frac{\partial l(\theta)}{\partial \beta 2}$.

Substituindo, na Equação (5.8), θ_i pela Equação (5.2), temos:

$$l(\theta) = \sum_{i=1}^{n} \delta_i \left\{ \sum_{h=1}^{H} w2_h \tanh\left(\sum_{p=1}^{P} w1_{hp} x_{pi} + \beta 1_h \right) + \beta 2 \right. \\ \left. - \log\left[\sum_{l \in C_i} \exp\left(\sum_{h=1}^{H} w2_h \tanh\left(\sum_{p=1}^{P} w1_{hp} x_{pl} + \beta 1_h \right) + \beta 2 \right) \right] \right\} \tag{5.11}$$

Derivando, agora, $l(\theta)$, em relação à $w1_{hp}$ temos:

$$\frac{\partial l(\theta)}{\partial W1_{hp}} = \sum_{i=1}^{n} \delta_i \left\{ \sum_{h=1}^{H} w2_h \sec h^2\left(\sum_{p=1}^{P} w1_{hp} x_{pi} + \beta 1_h \right) x_{pi} - \right. \\ \left[\sum_{l \in C_i} \exp\left(\sum_{h=1}^{H} w2_h \tanh\left(\sum_{p=1}^{P} w1_{hp} x_{pl} + \beta 1_h \right) + \beta 2 \right) \right]^{-1} \cdot \\ \left[\sum_{l \in C_i} \exp\left(\sum_{h=1}^{H} w2_h \tanh\left(\sum_{p=1}^{P} w1_{hp} x_{pl} + \beta 1_h \right) + \beta 2 \right) \cdot \right. \\ \left. \left. w2_h \sec h^2\left(\sum_{p=1}^{P} w1_{hp} x_{pl} + \beta 1_h \right) x_{pl} \right] \right\} \tag{5.12}$$

Ao derivarmos $l(\theta)$, em relação à $\beta 1_h$ temos:

$$\frac{\partial l(\theta)}{\partial \beta 1_h} = \sum_{i=1}^{n} \delta_i \left\{ \sum_{h=1}^{H} w2_h \sec h^2\left(\sum_{p=1}^{P} w1_{hp} x_{pi} + \beta 1_h \right) - \right. \\ \left[\sum_{l \in C_i} \exp\left(\sum_{h=1}^{H} w2_h \tanh\left(\sum_{p=1}^{P} w1_{hp} x_{pl} + \beta 1_h \right) + \beta 2 \right) \right]^{-1} \cdot \\ \left[\sum_{l \in C_i} \exp\left(\sum_{h=1}^{H} w2_h \tanh\left(\sum_{p=1}^{P} w1_{hp} x_{pl} + \beta 1_h \right) + \beta 2 \right) \cdot \right. \\ \left. \left. w2_h \sec h^2\left(\sum_{p=1}^{P} w1_{hp} x_{pl} + \beta 1_h \right) \right] \right\} \tag{5.13}$$

Derivando, agora, $l(\theta)$, em relação à $w2_h$ temos:

$$\frac{\partial l(\theta)}{\partial W2_h} = \sum_{i=1}^{n} \delta_i \left\{ \sum_{h=1}^{H} \tanh\left(\sum_{p=1}^{P} w1_{hp} x_{pi} + \beta 1_h \right) x_{pi} - \left[\sum_{l \in C_i} \exp\left(\sum_{h=1}^{H} w2_h \tanh\left(\sum_{p=1}^{P} w1_{hp} x_{pl} + \beta 1_h \right) + \beta 2 \right) \right]^{-1} \cdot \left[\sum_{l \in C_i} \exp\left(\sum_{h=1}^{H} w2_h \tanh\left(\sum_{p=1}^{P} w1_{hp} x_{pl} + \beta 1_h \right) + \beta 2 \right) \cdot \tanh\left(\sum_{p=1}^{P} w1_{hp} x_{pl} + \beta 1_h \right) \right] \right\} \quad (5.14)$$

Ao derivarmos $l(\theta)$, em relação à $\beta 2$ temos:

$$\frac{\partial l(\theta)}{\partial \beta 2} = \sum_{i=1}^{n} \delta_i \left\{ 1 - \left[\sum_{l \in C_i} \exp\left(\sum_{h=1}^{H} w2_h \tanh\left(\sum_{p=1}^{P} w1_{hp} x_{pl} + \beta 1_h \right) + \beta 2 \right) \right]^{-1} \cdot \left[\sum_{l \in C_i} \exp\left(\sum_{h=1}^{H} w2_h \tanh\left(\sum_{p=1}^{P} w1_{hp} x_{pl} + \beta 1_h \right) + \beta 2 \right) \right] .1 \right\}$$

$$\frac{\partial l(\theta)}{\partial \beta 2} = \sum_{i=1}^{n} \{1 - 1\}$$

$$\frac{\partial l(\theta)}{\partial \beta 2} = 0$$

5.2 Conclusão do Capítulo

Neste capítulo foram apresentados os componentes do algoritmo *backpropagation* para o ajuste das RNGs para a modelagem de dados, sejam dados onde a variável resposta pertença à família exponencial ou sejam dados de análise de sobrevivência.

No próximo capítulo serão apresentados alguns resultados de comparações empíricas entre as RNGs e os modelos lineares generalizados ou com o de regressão de Cox , utilizando para tanto base de dados reais. Além de alguns resultados comparativos entre os MLGs, as RNAs e as RNGs.

Capítulo 6

Metodologia e Resultados

Neste capítulo, a avaliação do potencial das Redes Neurais Generalizadas é realizada através de comparações empíricas com modelos convencionais, neste caso, com os Modelos Lineares Generalizados (MLGs) e com o Modelo de Regressão de Cox. Muitos problemas de predição de dados podem ser modelados através de MLGs. Deseja-se avaliar a capacidade das RNGs de retornar valores preditos em relação aos encontrados com os modelos convencionais. Sabe-se que a escolha da função de ligação influi no resultado do modelo e, portanto, a melhor alternativa consiste em comparar o modelo de RNGs com diversos MLGs, cada qual com uma função de ligação distinta. Também é comparada a resposta da RNG com a do Modelo de Cox. O modelo a ser escolhido será aquele que retornar o menor valor da *deviance* (desvio), e no caso da comparação com o modelo de Cox, aquele que retornar o maior valor do logaritmo da verossimilhança parcial.

No caso dos modelos RNGs os valores da *deviance* e da log-verossimilhança são calculados via Validação Cruzada. Ao utilizarmos a validação cruzada, a base de dados é subdividida aleatoriamente em três grupos, o primeiro (60% da total de observações) é utilizado para o treinamento e/ou ajuste dos parâmetros da rede e o segundo (20%) é utilizado como critério de parada do processo de treinamento e/ou escolha da solução final e o terceiro conjunto, denominado conjunto de teste, é utilizado para a verificação do processo. Durante o treinamento da rede, a *deviance* (ou a log-verossimilhança) é avaliada a cada atualização do vetor de pesos. Simultaneamente, o seu valor também é calculado para o conjunto de validação. O vetor de parâmetros final da rede corresponde àquele que representa o menor valor da *deviance* de validação durante a fase de treinamento (ou maior da log-verossimilhança, no caso do Modelo de Cox). Com base neste vetor de parâmetros calcula-se a *deviance* (ou a log-verossimilhança) para o

conjunto de teste. A Figura (6.5) ilustra o processo de treinamento e validação cruzada para a escolha do vetor final de parâmetros.

Para realizar uma comparação preliminar empírica, foram utilizadas 8 bases de dados sendo 4 bases reais e 4 bases simuladas. Para os estudos simulados comparou-se o desempenho das RNGs em relação às redes neurais artificiais e com os modelos lineares generalizados. As duas primeiras bases de dados teóricas apresentam variável resposta poisson enquanto que as demais, binomial. Para os estudos feitos com as bases de dados simulados todas as redes neurais (normais e generalizadas) possuem 10 nodos na camada intermediária, uma taxa de aprendizado de 0,00002 e realizavam no máximo 150000 iterações para cada reamostra.

A primeira base de dados teórica possui duas variáveis uma preditora *X* uniformemente distribuída no intervalo (0,10), $X \sim U(0{,}10)$, e a variável resposta *Y* com distribuição poisson com média *X*, $Y \sim P(X)$. A segunda base de dados teórica também possui 2 variáveis, sendo a variável preditora $X \sim U(1;1{,}6\pi)$ e a variável resposta $Y \sim P(\mu)$, onde $\mu_i = 5 \cdot [sen(x_i) + 1.01]$, $i = 1,2,\cdots,n$ e n é o número de registros do banco de dados. A terceira e a quarta base de dados teórico é baseado no exemplo extraído do Relatório Técnico: *LASSO – Patternsearch Algorithm with Application to Ophthalmology Data* (Shi,W. *et al., 2006)*. Sendo que a terceira deles possui 5 variáveis explicativas, sendo 4 delas (*B*1, *B*2, *B*3 e *B*4) com distribuição binomial com parâmetros $n = 1$ e $p = 0.5$, $B(1;0{,}5)$, e a outra denominada Ruído (R) como sendo $R = X + T + Z$, onde $X \sim U(0{,}1)$, $T \sim B(1;0{,}5)$ e $Z \sim N(0{,}1)$, e a variável resposta $Y \sim B(1,\pi)$, onde $\pi = \dfrac{e^{\mu}}{1+e^{\mu}}$, $\mu = -2 + 1{,}5B1 + 1{,}5B2 + 1{,}5B3 + 2B4 + R$. A última base de dados teórico apresenta 6 variáveis explicativas, as 5 primeiras idênticas às da base anterior e a sexta denominada $X1 \sim U(-2{,}2)$. A variável resposta *Y* tem a mesma forma da do problema anterior, porém, a forma de cálculo de μ neste exemplo é da seguinte forma: $\mu = -4(X1)^2 + 1{,}5B1 + 1{,}5B2 + 1{,}5B3 + 2B4 + R$.

As bases de dados simulados descritos neste parágrafo serão denominados, respectivamente, por BD1, BD2, BD3 e BD4. Sendo que as Bases de Dados BD3 e BD4 são compostas por um banco de dados com 1000 registros, enquanto que as outras duas bases (BD1 e BD2) são compostas de três bancos de dados de tamanhos diferentes (100, 200 e 1000 registros).

Podemos observar nas Figuras 6.1 à 6.4 e pelas Tabelas 6.1 e 6.2 que a redes nos retorna um valor da função desvio menor do que os modelos lineares generalizados.

As Tabelas 6.1 e 6.2 foram construídas a partir de 50 reamostras aleatórias distintas das bases de dados BD1 e BD2, respectivamente, para uma amostra de tamanho 100.

Figura 6.1 – Gráfico comparativo sobre o ajuste dos modelos preditivos ao referencial teórico dos dados do BD1 com amostras de tamanho 100 e 200

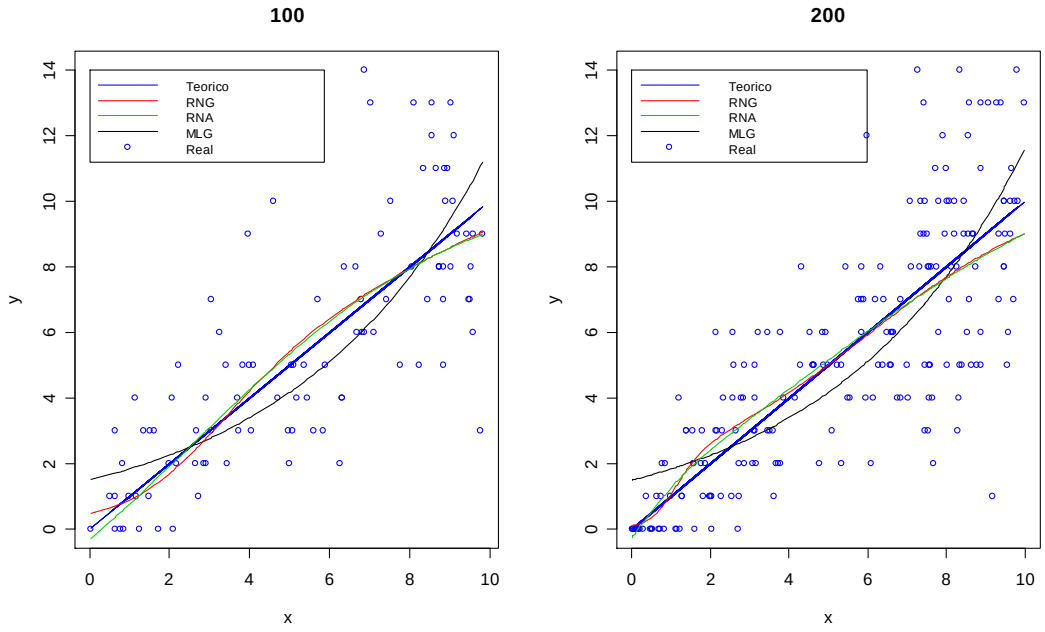
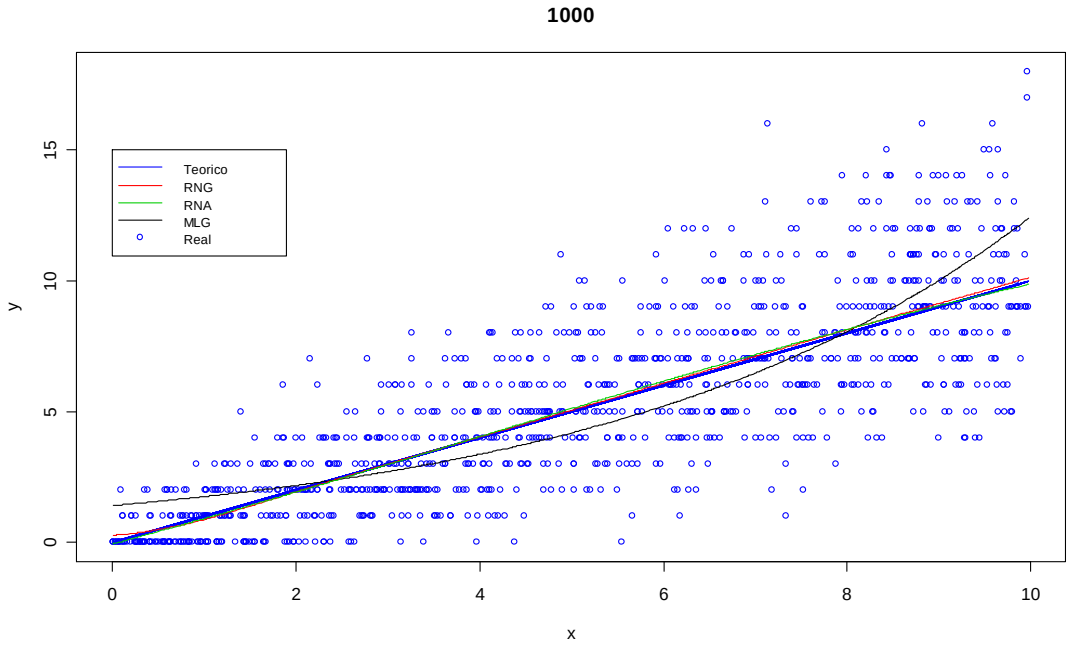


Figura 6.2 – Gráfico comparativo sobre o ajuste dos modelos preditivos ao referencial teórico dos dados do BD1 para uma amostra de tamanho 1000



Podemos notar pelas Figuras 6.1 e 6.2, ou seja, no caso em que $\mu_Y = x$, que as redes neurais se ajustam melhor ao modelo do que o MLG, o que pode ser notado visivelmente, pois as curvas dos valores ajustados pelas redes neurais (as curvas vermelha e verde) estão mais próximas da curva azul (a curva do modelo teórico) do que a curva preta ajustada por um modelo linear generalizado. Esta proximidade é melhor identificada à medida que o tamanho da amostra cresce. Na Figura 6.2 podemos ver que as curvas obtidas pelos ajustes dos modelos neurais caem praticamente em cima do modelo teórico, enquanto que a curva do modelo linear generalizado pouco se ajusta à curva teórica. A curvatura do modelo linear apresentada nos gráficos das Figura 6.1 à 6.4 se deve ao fato deste modelo tentar ajustar uma curva exponencial aos dados, pois está utilizando uma ligação canônica, ou seja, neste caso, temos que $\mu = e^{x\beta}$.

Vimos em nosso primeiro estudo que quando a função objetivo a ser estimada é uma função linear as redes neurais conseguiram resultados gráficos de ajuste melhores do que os dos modelos lineares generalizados. Por isso introduziremos agora uma não-linearidade aos dados e veremos se a redes neurais generalizadas e artificiais se ajustam tão bem quanto um MLG. Neste caso, a variável resposta $Y \sim P(\mu)$, onde $\mu_i = 5 \cdot [sen(x_i) + 1.01]$, $i = 1,2,\cdots,n$.

Figura 6.3 – Gráfico comparativo sobre o ajuste dos modelos preditivos ao referencial teórico dos dados do BD2 com amostras de tamanho 100 e 200

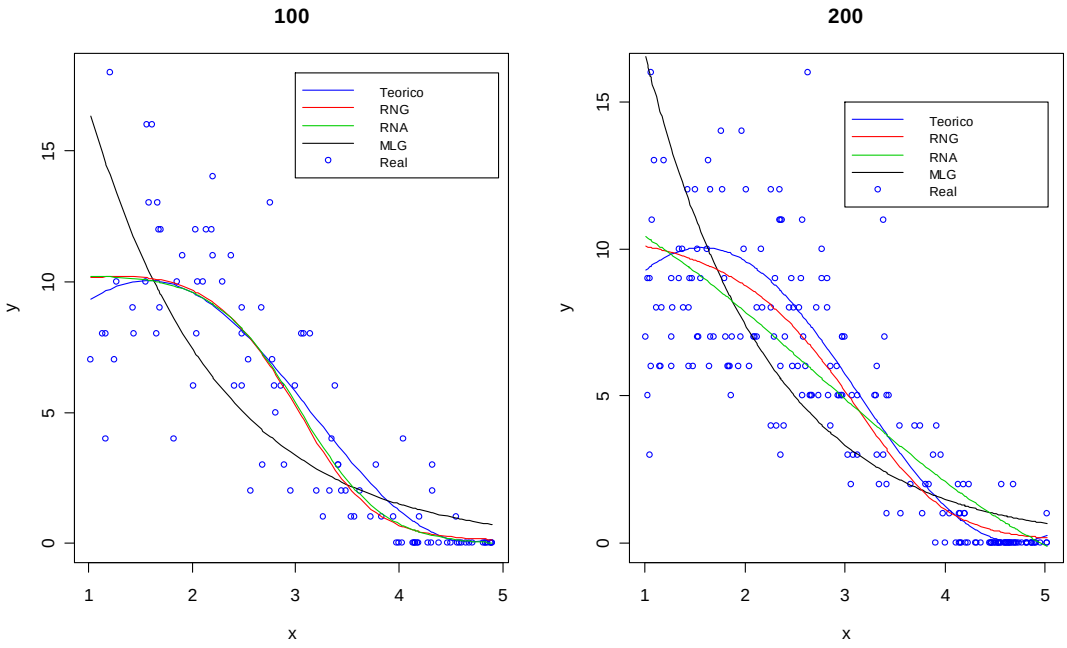
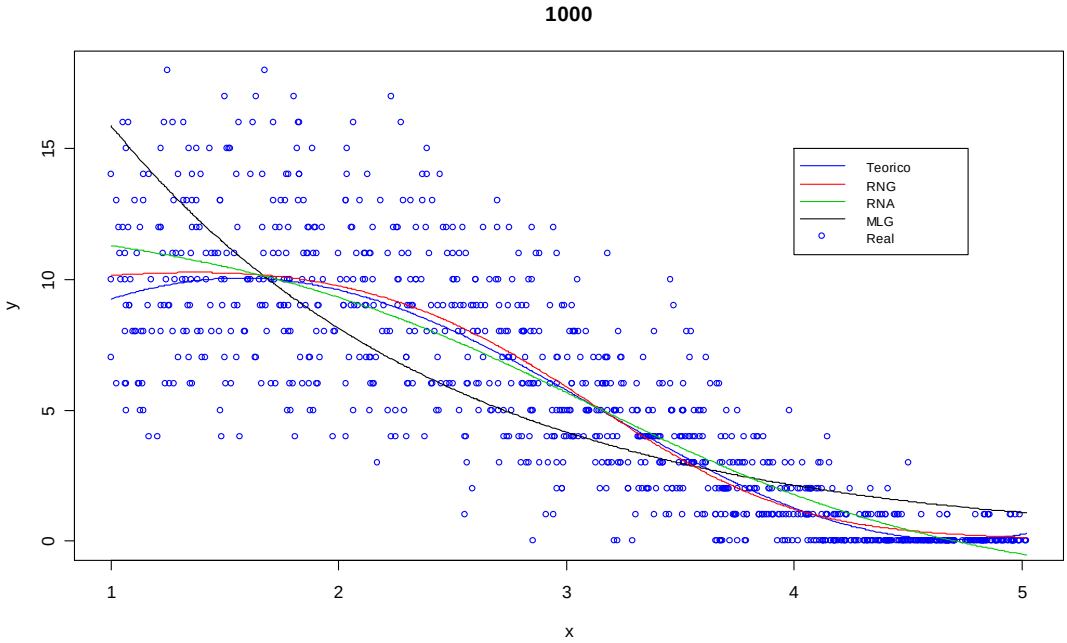


Figura 6.4 – Gráfico comparativo sobre o ajuste dos modelos preditivos ao referencial teórico dos dados do BD2 para uma amostra de tamanho 1000



Podemos notar pelas Figuras 6.3 e 6.4 que a redes neurais se ajustam melhor ao modelo do que o MLG, pois neste caso há uma não-linearidade muito mais latente do que no caso anterior, o que faz com que o modelo linear não consiga se ajustar perfeitamente à rede, o que se pode notar facilmente, pois este produz uma curva a qual não apresenta a curvatura inicial e final do modelo teórico.

Nota-se também que à medida que o tamanho da amostra cresce a RNG e a RNA (MLP *backpropagation* padrão) produzem resultados muito próximos. Porém, a RNG incorpora mais informações ao modelo (a distribuição da variável aleatória), o que não ocorre com a RNA, como, por exemplo, a capacidade de produzir soluções absolutamente positivas.

Comparamos os resultados das funções desvios para cada modelo e pudemos notar pelos dados apresentados nas Tabelas 6.1 e 6.2 que a RNG apresenta um desvio de validação menor ou igual ao das RNAs, ou seja, as RNGs retornam valores preditos mais próximos do referencial teórico do que as RNAs.

Tabela 6.1 – Tabela comparativa dos desvios dos modelos preditivos para o BD1 para uma amostra de tamanho 100

Modelo	Desvio	Média	Mediana	Desvio Padrão
RNG	Treinamento	60,7	60,8	8,78
	Validação	19,7	16,7	7,18
	Teste	21,8	20,1	8,03
	Total	102,2	102,3	2,03
RNA	Treinamento	61,7	64,9	8,49
	Validação	19,5	17,0	6,18
	Teste	22,1	20,9	7,70
	Total	103,3	102,7	2,67
MLG	Treinamento	67,5	68,7	9,54
	Validação	22,8	21,6	8,42
	Teste	22,2	19,3	8,73
	Total	112,5	112,3	0,71

Tabela 6.2 – Tabela comparativa dos desvios dos modelos preditivos para o BD2 para uma amostra de tamanho 100

Modelo	Desvio	Média	Mediana	Desvio Padrão
RNG	Treinamento	50,9	54,0	6,47
	Validação	15,4	14,3	3,36
	Teste	20,2	18,4	7,33
	Total	86,5	86,0	1,35
RNA	Treinamento	50,5	53,6	10,84
	Validação	15,3	14,8	4,14
	Teste	20,1	17,6	8,11
	Total	85,9	80,7	9,00
MLG	Treinamento	64,1	63,3	9,75
	Validação	20,0	19,3	8,92
	Teste	29,1	29,1	8,22
	Total	113,1	112,6	1,36

Com relação à modelagem preditiva para os exemplos teóricos onde a variável resposta é uma binomial, observamos que a RNG apresentou valores de *deviance* semelhantes e/ou menores do que os modelos lineares generalizados, ou seja, ela retornou valores preditos mais fiéis à realidade do que o MLG. A Tabela 6.4, para os dados onde há a presença de não-linearidade a rede neural generalizada apresenta uma *deviance* menor do que o modelo linear.

As Tabelas 6.3 e 6.4 foram construídas a partir de 100 reamostras aleatórias distintas das bases de dados BD3 e BD4, respectivamente, para uma amostra de tamanho 1000.

Tabela 6.3 – Tabela comparativa dos desvios dos modelos preditivos para o BD3 para uma amostra de tamanho 1000

Modelo	Função	Desvio	Média	Mediana	Desvio Padrão
RNG	Desvio	Treinamento	276,1	279,4	16,40
		Validação	99,9	102,3	14,59
		Teste	99,4	102,6	15,73
		Total	475,4	471,0	11,78
	SQR	Treinamento	41,7	41,2	2,63
		Validação	14,9	15,4	2,5
		Teste	14,9	15,1	2,58
		Total	71,5	71,1	1,54
RNA	Desvio	Treinamento	502,6	502,0	8,44
		Validação	168,4	168,9	6,52
		Teste	168,0	167,8	6,82
		Total	839,0	837,7	4,40
	SQR	Treinamento	44,8	44,9	2,86
		Validação	15,6	16,0	2,44
		Teste	21,3	20,8	2,26
		Total	81,6	81,9	2,09
MLG	Desvio	Treinamento	272,6	278,3	17,93
		Validação	100,6	102,1	14,98
		Teste	98,1	103,1	15,43
		Total	471,3	470,2	3,27
	SQR	Treinamento	80,8	79,5	9,64
		Validação	29,0	29,5	4,39
		Teste	28,4	28,0	5,02
		Total	138,1	137,0	9,38

Tabela 6.4–Tabela comparativa dos desvios dos modelos preditivos para o BD4 para uma amostra de tamanho 1000

Modelo	Função	Desvio	Média	Mediana	Desvio Padrão
RNG	Desvio	Treinamento	305,8	311,1	18,62
		Validação	110,6	108,4	15,03
		Teste	102,1	98,8	10,83
		Total	518,6	518,2	5,79
	SQR	Treinamento	44,5	45,0	3,36
		Validação	16,2	15,6	2,46
		Teste	15,0	14,4	1,96
		Total	75,6	75,6	0,69
RNA	Desvio	Treinamento	844,1	857,7	19,86
		Validação	284,8	288,5	17,42
		Teste	281,2	284,7	55,46
		Total	1410,1	1435,0	88,78
	SQR	Treinamento	131,2	140,2	30,31
		Validação	45,2	47,8	9,34
		Teste	56,3	51,8	13,04
		Total	232,7	239,6	26,78
MLG	Desvio	Treinamento	793,0	791,7	5,91
		Validação	268,3	268,5	8,08
		Teste	261,1	263,0	6,67
		Total	1322,4	1323,7	1,58
	SQR	Treinamento	278,4	277,5	12,91
		Validação	95,5	95,5	5,10
		Teste	93,0	93,5	7,04
		Total	466,9	467,0	17,77

Podemos observar que os valores da *deviance* para a RNA são bem superiores aos da RNG isso se deve ao fato de que a função a ser minimizada nas redes normais não é a log-verossimilhança da variável resposta e sim a Soma dos Quadrados dos Resíduos (SQR), ou seja, o valor da *deviance* neste caso é superior ao encontrado para as RNG. Já a razão de a RNG apresentar um valor de SQR menor do que a RNA é devido ao fato de que como a RNG maximiza a verossimilhança, os valores preditos por ela são mais próximos da realidade do que os encontrados via redes normais, e como a SQR é formada por uma soma quadrática das diferenças entre os valores reais e os preditos temos que a RNG consegue valores de SQR melhores do que os encontrados via RNA.

Vamos realizar agora um estudo comparativo empírico entre as RNGs e os MLGs com dados de 4 bancos de dados reais. São eles:

- A primeira representa um estudo desenvolvido pela Dra. Jane Brockmann (Brockmann, 1996), do Departamento de Zoologia da Universidade da Flórida. No estudo em questão, deseja-se prever o número de satélites (ou seja, número de caranguejos machos que rodeiam uma fêmea) de acordo com as características dos 173 caranguejos fêmeas, são elas: cor, estado da espinha dorsal, peso e comprimento da carapaça, sendo que as duas primeiras variáveis são qualitativas e as outras 2 quantitativas. A variável resposta, o número de satélites, neste caso segue a distribuição de Poisson.

- A segunda base de dados representa uma pesquisa de marketing de uma determinada companhia telefônica dos Estados Unidos e apresentam informação de 1.000 domicílios e suas respectivas famílias. Porém como para muitas destas entrevistas apresentaram dados faltantes para uma ou mais variáveis, tais registros foram excluídos restando um total de 757. Foram observadas 10 variáveis, dentre elas apenas duas delas eram quantitativas (nº de vezes em que o indivíduo se mudou nos últimos 10 anos e o uso médio mensal do domicílio). Os dados foram obtidos através de uma pesquisa realizada por telefone (Watson, 1982). A variável de interesse neste estudo, que é a preferência por determinada operadora de telefonia segue a distribuição Binomial.

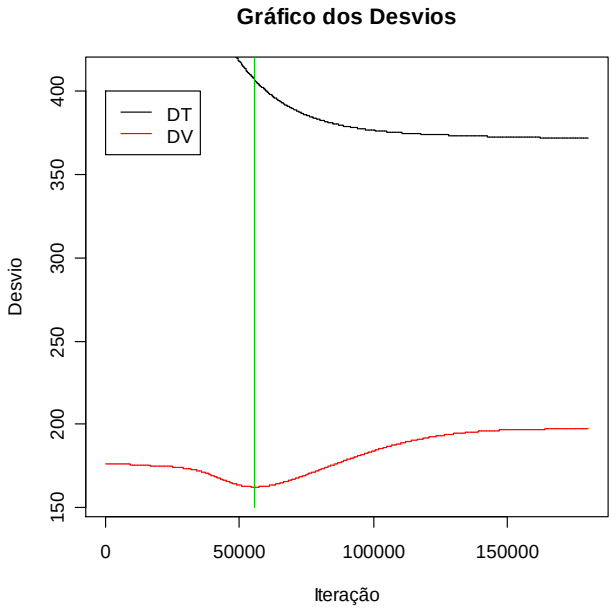
- A terceira base de dados é sobre a avaliação do preço de imóveis nos EUA na cidade de Boston e apresenta informações relevantes sobre a localização deste como, por exemplo, o índice de criminalidade do distrito, o percentual de negros e de pobres na região, a proximidade de rio, comércio, a fluidez do trânsito dentre outras. Das 14 variáveis que compõem esta base somente 2 são qualitativas, o índice de acessibilidade às principais vias de trânsito e se o imóvel se localiza perto do Rio Charles. A suposição inicial para a variável de interesse, o preço do imóvel em milhares de dólares, é que a mesma segue a distribuição Gama.

- E por último uma base de dados de sobrevivência de 128 pacientes acometidos pela doença granulomatosa crônica. Estes dados encontram-se no livro *Modelling Survival Data in Medical Research (Collett, 1994)* e possui 12 variáveis sendo 4 quantitativas e 8 qualitativas.

O modelo de RNGs para a distribuição Poisson foi comparado com o modelo linear generalizado com função de ligação logarítmica. O modelo de RNGs para Binomial foi comparado com 3 outros modelos lineares generalizados: um com função de ligação logística, outro com função *probit* e o terceiro com função complemento *log-log*. O modelo de rede neural generalizada para Gama foi comparado com o modelo linear generalizado com função de ligação inversa. O modelo de rede neural generalizada para Cox foi comparado com o modelo de regressão de Cox. Os resultados das comparações para a base de dados *Caranguejo (Poisson), Companhia Telefônica (Binomial), Comércio Imobiliário (Gama) e Análise de Sobrevivência (Cox)* estão descritos a seguir. As RNGs utilizadas nos estudos empíricos realizados com dados reais tinham 10 nodos na camada escondida, uma taxa de aprendizado de 0,00002 e realizavam no máximo 150000 iterações para cada reamostra (exceto à rede aplicada ao banco *Comércio Imobiliário* onde foram realizadas 180000 iterações)

a) POISSON

Figura 6.5 - Gráfico de comportamento da *deviance* para os conjuntos de treinamento e validação referente à base de dados *Caranguejo*.



A Figura 6.5 mostra o número de iterações e as curvas de deviance para uma amostra do conjuntos de treinamento e validação para a base de dados *Caranguejo*. A

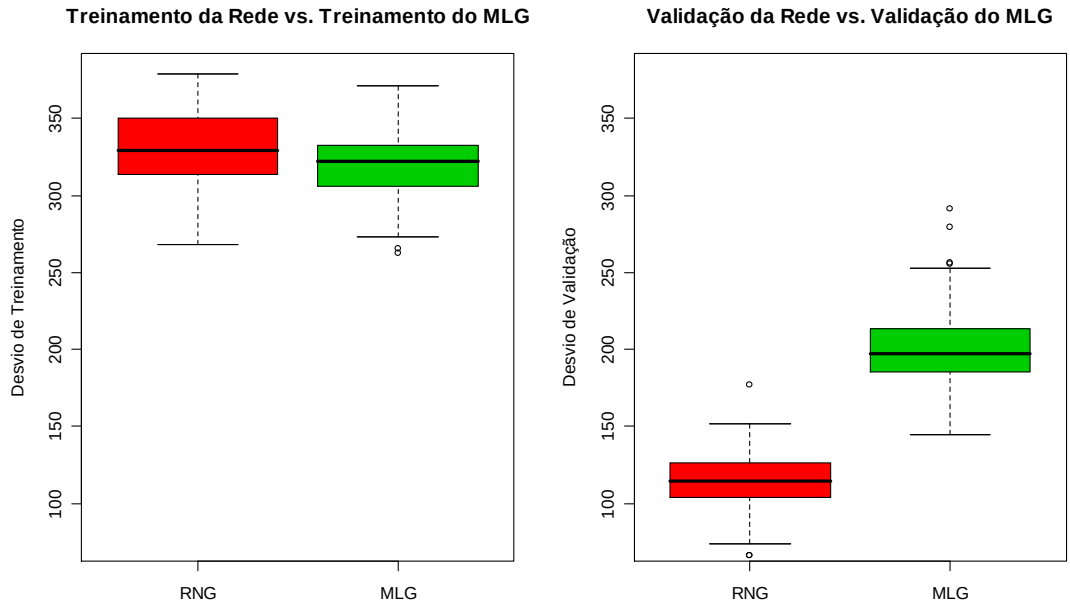
linha verde marca a iteração onde ocorreu a *deviance* mínima de validação, representando o ponto no qual foram definidos os parâmetros da rede.

De acordo com a Figura 6.5 não se faz necessário realizar todas as 150 mil interações, pois com aproximadamente 55 mil o algoritmo encontrou-se a *deviance* de validação mínima, ou seja, ele encontrou o vetor de parâmetros da rede que retorna o menor erro de predição.

Tabela 6.5 - Resultados da *deviance* para a base *Caranguejo* utilizando a verossimilhança de Poisson

Modelo	Desvio	Média	Mediana	Desvio Padrão
RNG	Treinamento	331,2	329,5	24,48
	Validação	115,2	114,7	19,82
	Teste	139,2	138,5	22,36
	Total	585,6	584,1	19,07
MLG	Treinamento	319,5	322,1	22,95
	Validação	201,7	197,2	26,02
	Teste	202,5	196,3	26,45
	Total	723,7	729,3	50,44

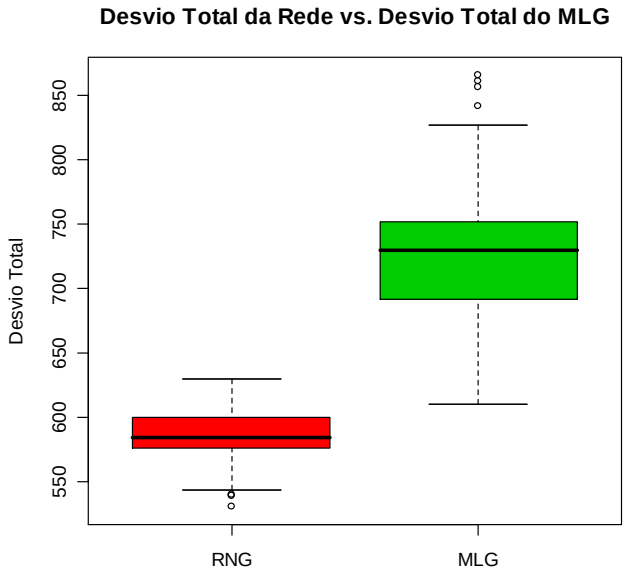
Figura 6.6 - Gráfico comparativo do comportamento dos desvios de Treinamento e Validação entre a RNG e o MLG referente à base C*aranguejo*



Podemos observar pelos dados apresentados na Tabela 6.5 e visualmente na Figura 6.6 que mesmo o Modelo Linear Generalizado se ajustando melhor aos dados (amostra de treinamento) do que a Rede Neural Generalizada, esta possui resultados de

previsão melhores do que o Modelo Linear Generalizado para os dados da base *Caranguejo*.

Figura 6.7 - Gráfico comparativo do comportamento do Desvio Total entre a RNG e o MLG referente à base C*aranguejo*



Na Figura 6.7 podemos observar que a *deviance* total (soma das *deviances* de treinamento, validação e de teste) da RNG é menor do que a do MLG, fato este que fornece evidência da existência de interação não-aditiva ou de correlação não-linear entre as variáveis preditoras, entretanto, essa característica não pode ser quantificada em função da sua representação interna na topologia da Rede Neural, o que faz dela um modelo preditivo.

b) BINOMIAL

Podemos observar pela Tabela 6.6 que a RNG modela melhor os dados provenientes da base *Companhia Telefônica* a qual tem variável resposta regida pela distribuição binomial. Observamos também que os resultados da função desvio entre os vários modelos são muito próximos, o que evidencia a não existência de não-linearidade dos dados, o contrario do observado com relação à base de dados *Caranguejo* onde as *deviances* totais dos modelos são muito distintas, sendo a obtida pela rede neural generalizada bem menor do que a conseguida via modelo linear generalizado.

Tabela 6.6. Resultados da *deviance* para a base *Companhia Telefônica* utilizando a verossimilhança da Binomial.

Modelo	Desvio	Média	Mediana	Desvio Padrão
RNG	Treinamento	581,1	580,2	11,93
	Validação	199,5	200,0	5,84
	Teste	201,7	200,8	8,97
	Total	982,4	979,1	14,50
MLG Logit	Treinamento	559,1	560,8	9,71
	Validação	216,1	211,7	18,59
	Teste	216,0	212,6	19,54
	Total	991,2	987,8	23,77
MLG Probit	Treinamento	559,5	561,1	9,86
	Validação	215,8	212,3	17,72
	Teste	216,0	212,3	19,36
	Total	991,3	988,6	22,43
MLG Complemento Log-log	Treinamento	559,7	560,9	10,25
	Validação	223,1	214,3	24,35
	Teste	224,6	214,0	29,47
	Total	1007,0	1009,0	32,84

c) GAMA

Tabela 6.7 - Resultados da d*eviance* para a base *Comércio Imobiliário* utilizando a verossimilhança da Gama

Modelo	Desvio	Média	Mediana	Desvio Padrão
RNG	Treinamento	10,29	10,33	0,73
	Validação	3,52	3,48	0,74
	Teste	3,95	3,70	0,95
	Total	17,76	17,56	0,80
MLG	Treinamento	10,03	10,14	0,75
	Validação	3,58	3,56	0,75
	Teste	3,83	3,66	0,83
	Total	17,44	17,40	0,22

Na Tabela 6.7 podemos observar que as *deviances* de treinamento e validação da RNG são muito próximas daquelas calculadas para Modelo Linear Generalizado, ou seja, para este conjunto de dados, *Comércio Imobiliário*, tanto a RNG como o MLG apresentaram desempenhos parecidos, pois a diferença entre elas foi muito pequena.

d) ANÁLISE DE SOBREVIVÊNCIA (COX)

Figura 6.8 - Gráfico comparativo do comportamento das Log-verossimilhanças de Treinamento e Validação entre a RNG e o Modelo de Cox referente à base de dados *Análise de Sobrevivência*

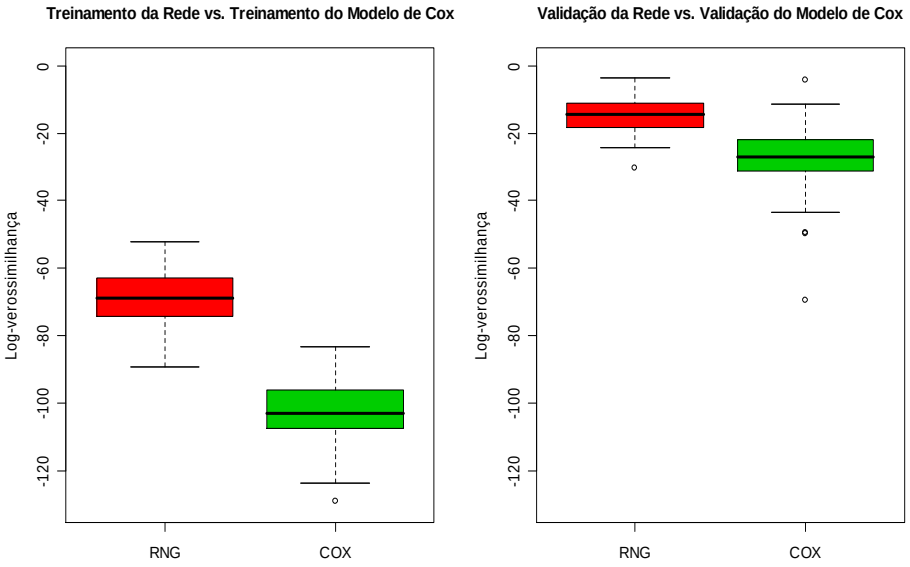
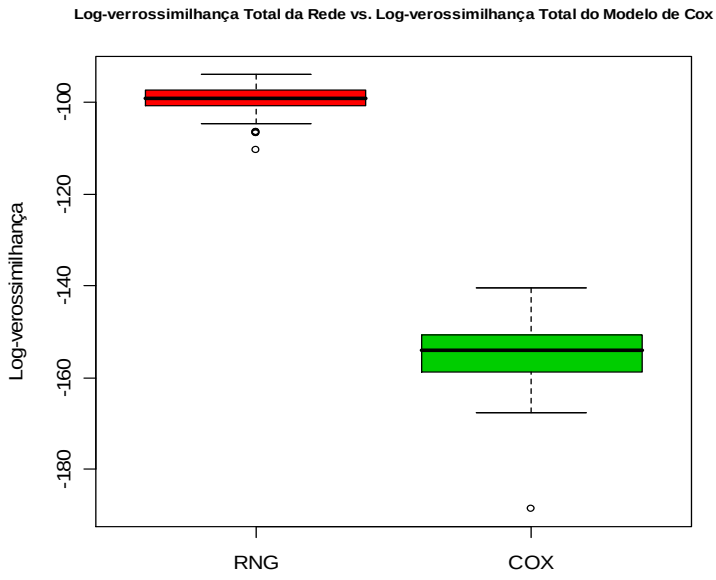


Figura 6.9 - Gráfico comparativo do comportamento da Log-verossimilhança Total entre a RNG e o Modelo de Cox referente à base de dados *Análise de Sobrevivência*



Nas Figuras 6.8 e 6.9 podemos observar que a log-verossimilhança da RNG é maior do que a do Modelo de Cox, este fato fornece evidência de um melhor ajuste da rede aos dados do banco de dados *Análise de Sobrevivência* e uma melhor previsibilidade da rede em relação ao modelo convencional

Tabela 6.8 - Resultados da Log-verossimilhança para a Base de dados *Análise de Sobrevivência* utilizando a verossimilhança-parcial do Modelo de Cox.

Modelo	Desvio	Média	Mediana	Desvio Padrão
RNG	Treinamento	-68,89	-68,99	7,9
	Validação	-14,80	-14,53	4,84
	Teste	-15,73	-15,59	4,50
	Total	-99,41	-99,13	2,95
COX	Treinamento	-102,80	-103,00	9,18
	Validação	-27,07	-26,89	9,08
	Teste	-25,24	-26,02	7,74
	Total	-155,10	-154,10	7,02

Na Tabela 6.8 podemos sentir o quão melhores são as previsões via RNG em relação ao modelo de Cox, já que neste caso todas as log-verossimilhanças foram bem maiores cerca de 30% a 40%.

6.1 Conclusão do Capítulo

Os resultados obtidos pelas RNGs para as bases de dados *Caranguejo* e *Análise de Sobrevivência* caracterizam-se por um desempenho de previsão melhor do que os Modelos Lineares Generalizados para as mesmas bases, respectivamente. Este fato fornece evidência da existência de uma interação não-aditiva significativa entre as varáveis preditoras ou uma correlação não-linear entre as mesmas. Entretanto, essa característica não pode ser quantificada em função da sua representação interna na topologia da Rede Neural.

Já os resultados obtidos pela rede para as bases de dados *Comércio Imobiliário* e *Companhia Telefônica*, a RNG e o MLG obtiveram resultados de *deviance* muito próximos, o que indica que não deve haver neste caso nenhuma não-linearidade. Porém, apesar de resultados totais próximos, o valor da *deviance* de validação para a RNG continua menor, ou seja, ela continua produzindo valores preditos mais fiéis à realidade.

Podemos notar que a Rede Neural Generalizada foi um modelo capaz de predizer bem nas diversas situações nas quais ele foi testado, produzindo nestes casos

valores preditos melhores do que os demais modelos: Modelos Lineares Generalizados e Redes Neurais Artificiais.

Estes resultados se devem a fato de que as RNGs conseguem unir uma forma de modelagem computacional não-paramétrica com a informação da verossimilhança. Com mais informações o aprendizado da rede neural melhora o que resulta em predições mais condizentes à realidade em estudo.

Capítulo 7

Conclusões e Trabalhos Futuros

Este trabalho apresentou uma descrição e análise dos modelos de Redes Neurais Generalizada, as quais se mostraram bastantes eficientes no que se refere à predição de dados visto que em todas as simulações e comparações empíricas com dados reais ela obteve sempre o menor desvio de validação e/ou o maior valor da log-verossimilhança parcial (quando comparada ao Modelo de Cox).

. Os resultados obtidos através da comparação das RNGs com os MLGs e o Modelo de Cox aplicados às oito bases de dados demonstram a capacidade do modelo neural generalizado de ajustar automaticamente seus parâmetros com base em um conjunto de observações sem utilização de informações prévias sobre a correlação das variáveis de interesse, gerando modelos de predição com desempenho no mínimo comparáveis aos modelos tradicionais.

Esta característica evidencia os ganhos obtidos a partir da integração dos modelos estatísticos tradicionais com os modelos computacionais neurais (RNA). Esta integração vem sendo realizada em muitos trabalhos, principalmente na área da saúde (Biganzoli *et al*, 1998 e 2002 e Foraggi e Simon, 1995), onde são utilizados freqüentemente os modelos de Cox. Porém, apesar de utilizarmos um outro algoritmo de treinamento (*backpropagatio*) do que o utilizado pelos autores anteriormente citados (Newton-Raphson), esta é a primeira vez em que são demonstradas as equações de ajuste dos pesos da rede.

Diferentemente dos resultados obtidos via *PLANN – Partial-Logistic Artificial Neural Networks* criado por Biganzoli e seus colegas em 1998, as RNGs obtiveram resultados bastantes satisfatórios, ou seja, conseguiu produzir valores preditivos melhores do que os modelos tradicionais (MLG e Modelos de Regressão de Cox) e com um custo computacional relativamente baixo.

Vale a pena salientar que os modelos de Redes Neurais Generalizadas são modelos preditivos e não explicativos. Ou seja, neste caso não se pode dizer o quanto a variável resposta cresceria ou diminuiria com o incremento de uma unidade em uma determinada variável explicativa. Tais modelos são muito úteis principalmente quando se quer estimar valores de variáveis importantes a algum estudo omitidos por informantes, como a renda de uma pessoa, a produtividade de uma empresa, *etc*.

Um exemplo de uma situação onde os modelos preditivos são requeridos é na estimação dos valores das diversas rendas que uma pessoa pode ter e não informa ao entrevistador censitário do IBGE. Neste caso, é preferível um modelo que informe o quanto este indivíduo recebeu do que um que me diga quanto um homem ganha mais ou menos do que uma mulher, quanto um branco ganha mais ou menos do que um amarelo. Isso não quer dizer que devemos deixar de lado os modelos explicativos, mas que existem situações em que tais explicações podem ficar para segundo plano.

Uma questão a qual gostaria de levantar é sobre a previsão intervalar. Nós não nos preocupamos muito com isso durante o processo de criação das RNGs, porém gostaria de deixar uma sugestão de previsão intervalar para a estimativa média. Ao repetirmos o processo de estimativas dos parâmetros da rede *k* vezes para a mesma amostra, obteremos *k* valores preditos para cada registro. Gerando desta forma, para cada registro, uma estimativa média e uma variabilidade desta estimativa média.

Com relação aos trabalhos que pretendo realizar em um futuro próximo estão:

- Aplicação de Pruning para a simplificação do modelo neural (Costa *et al.*,2003);
- Utilização de Algoritmos mais Eficientes (mais rápidos) e mais Robustos (menos sensíveis a mínimos locais) para a atualização dos pesos da rede;
- Comparação das RNGs com outros métodos preditivos de imputação de dados reais utilizados pelo IBGE.

Bibliografia

Andersen, P.K. (1982) Testing Goodness of Fit for Cox's Regression and Life Model. Biometrics **38** 67-77

Andersen, P.K. e Gil, R (1982) Cox's Regression Model for Couting Process: A Large Sample Study. *Annals of Statistics* **10** 1100-1200

Bartlett, P.L. (1997) For valid generalization, the size of the weigths is more important than the size of the network. *Advances in Neural Information Processing Systems,* **9** 134-140.

Becker, R.A.; Chambers, J.M. e Wilks A.R.(1988) T*he new S Language: A programming environment for data analysis and graphics.* Wadsworth & Brooks, Pacific Grove, CA.

Biganzoli, E; Boracchi, P.; Mariani, L. e Marubini, E. (1998). Feed forward Neural Networks for the analysis of censored survival data: A partial logistic regression approach. *Statistics in Medicine*, **17** 1169-1186.

Biganzoli, E.; Boracchi, P.; Marubini, E. (2002) A general framework for neural network models on censored survival data .*Neural Networks*, **15** 209-218

Bland, R.G.; Goldfarb, D. e Todd, M.J. (1981) The Ellipsoid Method: A Survey. *Operations Research,* **29**(6) 1039-1091

Braga, A.P.; Carvalho, A.C.P.L.F; Ludermir, R.B. (2000) *Redes Neurais Artificiais Teoria e Aplicações.* LTC, Rio de Janeiro

Breslow, N. (1972) Contribuição à discussão do artigo de D.R.Cox. *Journal of the Real Statistics Society B,* **34** 216-217

Brockmann, H. J. (1996) Satellite male groups in horseshoe crabs *Limulus polyphemus*. *Ethology* **102** 1-21.

Collett, D. (1994) *Modelling Survival Data in Medical Research.* Chapman & Hall, London.

Colosimo, E.A. e Giolo, S.R. (2006) *Análise de Sobrevivência Aplicada.* Edgard Blücher, São Paulo.

Cordeiro, G.M. e Paula, G.A. (1992) Estimation, large-samples parametric tests and diagnostics for non-exponencial family nonlinear models. *Comunications in Statistics, Simulation and Computation,* **21** 149-172

Cordeiro, G.M.e Neto, E.A.L. (2004) *Modelos Paramétricos*. Livro texto de minicurso , XVI SINAPE, UNICAMP, Campinas, SP.

Costa, M. A. (2002) Controle por Modos Deslizantes da Generalização em Aprendizado de Redes Neurais Artificiais. PhD *thesis*, CPDEE, UFMG.

Costa, M.A.; Braga, A.P. e Menezes, B.R. (2003) Improving neural networks generalization with new constructive and pruning methods. *Journal of Intelligent & Fuzzy Systems*, **13** 75-83.

Costa, M.A.; Braga, A.P. e Menezes, B.R. (2006) Improving generalization of MLPs with sliding mode control and the levenberg-marquardt algorithm. *Neurocomputing*. (*in press*)

Cox, D.R. e Snell, E.J. (1968) A General Definition of Residuals. *Journal of the Real Statistics Society*, **30** 248-275

Cox, D.R. e Hinkley, D.V.(1974) *Theoretical Statistics*,Chapman and Hall, London

Cox, D.R. (1975) Partial Likelihood. *Biometrika,* **62** 269-276

Cox, D.R. (1979)A Note on the Graphical Analysis of Survival Data. *Biometrika,* **66** 188-190

Cybenko, G. (1988) Continuos valued neural networks with two hidden layers are sufficient. Technical report. Department of Computer Science, Tufts University.

Cybenko, G. (1989) Approximation by superpositions of a sigmoid function. *Mathematics of Control, Signals and Siystems*, **2** 303-314

Dugas, C; Bengio, Y; Chapados, N; Vicent, P; Denoncourt, G e Fournier, C (2003). Statistical Learning Algorithms Applied to Automobile Insurance Ratemaking. In: Jain, L e Shapiro, A.F., editor, *Intelligent Techniques in the Insurance Industry: Theory and Applications*. pp.1-31

Efron, B. (1977) The Efficiency of Cox's Likelihood for Censored Data. *Journal of the American Statistical Association,* **72** 557-565

Fahlman, S.E. (1988) An empirical study of learning speed in back-propagation networks. Technical Report, Carnegie Mellow University

Faraggi, D.; Simon R. (1995) A neural network model for survival data. *Statistics in Medicine* **14** 73-82

Ferreira, A. M e Molina, J.A.C. (2000) Introdução ao R. In. *Introdução à Linguagem R para Análise Estatítica*, Rio de Janeiro, IBGE: CDDI.

Goutte, C. (1997) Note on free lunches and cross-validation. *Neural Computation,* **9**(6) 1245-1249.

Haykin, S. (2001) *Redes neurais: Princípios e prática*. Bookman, Porto Alegre.

Hebb, D.O. (1949) *The Organization of Behavior: A Neuropschological Theory*. Wiley, Nova York.

Hopfield, J.J. (1982) Neural Networks and Phisical Sistems with Emergent Collective Computational Abilites. *Proceedings of the National Academy of Sciences,* **79** 2554-2558

Kalbfleish, J.D. e Prentice, R.L. (1980) *The Statistical Analysis of Failure Time Data.* John Wiley and Sons, New York.

Marquardt, D.W. (1963) An algorithm for least squares estimation of nonlinear parameters. *Journal of the Society for Industrial Applied Mathematics,* **11**(2) 431 - 441.

McCulloch, W.C. e Pitts, W. (1943) A Logical Calculus of the Ideas Immament in Nervous Activity. *Bulletin of Mathematical Biophisics,* **5** 115-133

Minsky, M.L. e Papert, S.A. (1969) *Perceptrons.* Mit Press, Cambridge.

Nelder, J. A. e Wedderburn, R.W.M. (1972). Generalized Linear Models. *Journal of the Royal Statistical Society A* **135**, 370-384.

Parma, G.G.; Menezes, B.R. e Braga, A. P. (1998) Sliding mode algorithm for training multilayer artificial neural networks. *Eletronics Letters*, **34**(1) 97-98.

Parma, G.G.; Menezes, B.R. e Braga, A. P. (1999) Neural networks learning with sliding mode control: The sliding mode backpropagation algorithm. *International Journal of Neural Systems,* **9**(3) 187-193

Paula, A.G. (2004) *Modelos de Regressão com apoio computacional*. USP, São Paulo

Peto, R. (1972) Contribuição à discussão do artigo de D.R.Cox. *Journal of the Real Statistics Society B,* **34** 205-207

Riedmiller, M. e Braun, H. (1993) A direct adaptive method for faster backpropagation learning: The Rprop algorithm. In. *Proceedings of the IEEE Intl. Conf. on Neural Networks*, pp 586-591, San Francisco, CA.

Rosenblatt, F. (1958) The Perceptron: A probabilistic model for information storage and organization in the brain. *Psicological Review,* **65** 386-408

Rumelhart, D.E.; Hilton, G.E. e Willians, R.J. (1986) Learning Internal Representations of Back-propagation Errors. In. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* pp. 318-362, Cambridge, MA:MIT Press.

Schoenfeld, D.A. (1980) ChiSquare goodness of Fit Tests for the Proportional Hazards Regression Models. *Biometrika,* ***6*****7** 145-153

Shi, W.; Wahba, G.; Wrigth, S.; Lee, K.; Klein, R. e Klein, B.(2006) LASSO – Patternsearch Algorithm with Application to Ophthalmology Data. Technical Report. Department of Statistics of the University of Wisconsin.

Shor, N.Z. (1977) Cut-off method with space extension in convex programming problems. *Cybernetcs*, **12** 94-96

Silva, F.M. e Almeida, L.B. (1990) Speeding up backpropagation. In. Ekmiller,R., editor, *Advanced Neural Computers,* pp. 151-158, Amsterdam, Elsevier North Holland.

Statistical Sciences (1993) *S-Plus Programmer's Manual, Version 3.2.* Seattle, StatSci, a division of MathSoft, Inc.

Stone, M. (1978) Cross-validation choice and assessment of statistical predictions. *Journal of Royal Statistical Society B,* **36** 111-147

Storer, B.E.; Wacholder, S. e Breslow, N.E. (1983). Maximum Likelihood Fitting of General Risk Models of Stratified Data. *Applied Statistics,* **32** 177-181

Struthers, C.A. e Kalbfleisch, J.D. (1986) Misspecific Proportional Hazards Models. *Biometrika,* **73** 363-369

Takahaxhi, R.H.C.; Peres, P.L.D e Ferreira, P.A.V. (1997) H2/H-infinity multiobjective PID design. *IEEE Control Systems Magazine,* **17**(5) 37-47.

Teixeira, R. A. (2001) T*reinamento de Redes Neurais Artificiais Através de Otimização Multi-Objetivo: Uma Nova Abordagem para o Equilíbrio entre a Polarização e a Variância.* PhD *thesis,* CPDEE, UFMG.

Teixeira, R. A.; Braga, A.P.; Takahaxhi, R.H.C. e Saldanha, R.R. (2001) Improving generalization of MLPs with multi-objective optimization. *Neurocomputin,* **35** (1 - 4) 189-194.

Tollenaere, T. (1990) SuperSAB: Fast adaptive back propagation with good scaling properties. *Neural Networks,* **3**(5) 561-573

Utkin, V.I. (1978) *Sliding modes and their application in variable structure systems.* MIR, Moscow.

Watson, J.W.(1982) In.Chambers, J.M. e Hastie, T.J., editores, (1992) Statistical Models in S. Wadsworth and Brooks, Pacific Grove, CA , pág. 49.

Wei, L.J. (1984) testing Goodness of Fit for Proportional Hazards model with Censored Observation. *Journal of the American Statistical Association,* **79** 649-652

Weigend, A.S.; Huberman, B.A. e Rumelhart, D.E. (1990) Predicting the Future: A Connectionist Approach. *International Journal of Neural Systems,* **1** 193-209

Widdrow, B. e Hoff, M.E. (1960) Adaptative Switching Circuits. *IRE WESCON Convention Record,* pp. 96-104

Anexo A

Principais Rotinas de R

Todos os programas foram construídos em R. O R é um *software* e um ambiente onde se pode tanto criar como utilizar as funções já nele existentes. A grande vantagem em se utilizar o R é que o mesmo é um *software* livre, ou seja, gratuito. A sua maior desvantagem é que o mesmo não faz o swap, ou seja, ele só utiliza a memória RAM que você possui. O que não foi problema, pois meu micro tinha bastante memória RAM, 1Gb, o que me possibilitou executar todas as rotinas em um mesmo micro, além disso, quanto mais memória, mais rápido as rotinas em R vão rodar.

Em relação ao tempo de execução dos programas, eles foram relativamente curtos, a rotina da RNG para Poisson, para um banco de dados com 10 variáveis e 173 registros executa aproximadamente 30.000 interações por minuto. Já para bancos de dados maiores como o utilizado para a rotina da RNG para Binomial, com 31 variáveis e 757 registros demora cerca de 10 minutos para executar 150.000 interações. No caso do programa para Gama o programa demora cerca de 15 minutos para rodar cerca de 100.000 interações, isto porque ele tem um teste de adequação a mais em sua rotina. Portanto, não há como determinar o tempo que cada rotina leva, pois isto depende tanto do número de interações que o programa terá de executar como do tamanho do banco de dados. Outro importante fato é que as variáveis categóricas para serem utilizadas deverão ser *k* transformadas em variáveis *dummy*.

Rotina de uma RNG para Poisson

```
bkp.mod.poi <- function (D,X,DV,XV,DT,XT,H,eta,int)
{

 ###   Leitura de Dados e Inicialização dos pesos da rede   ###

 X <- t(as.matrix(X)); D <- t(as.matrix(D)); n <- dim(X)[2]
 XV <- t(as.matrix(XV)); DV <- t(as.matrix(DV)); nv <- dim(XV)[2]
 XT <- t(as.matrix(XT)); DT <- t(as.matrix(DT)); nt <- dim(XT)[2]

 p <- dim(X)[1]
 w1 <- matrix((rnorm(H*p)*0.01),H,p)
 w2 <- matrix((rnorm(H)*0.01),1,H)
 b1 <- matrix((rnorm(H)*0.01),H,1)
 b2 <- matrix(log(mean(D)),1,1)
 eta <- eta

 ###   Criando vetores e variáveis auxiliares   ###

 um <- matrix(1,1,n)
 umv <- matrix(1,1,nv)
 umt <- matrix(1,1,nt)
 Desvio.T <- numeric(int)
 Desvio.V <- numeric(int)
 Norma.W <- numeric(int)

 pesos <- as.list(1:5)
 pesos[[1]] <- w1
 pesos[[2]] = w2
 pesos[[3]] = b1
 pesos[[4]] = b2
 pesos[[5]] = 1

 Result <- as.list(1:5)

 ###   Calculando a saída da rede - supondo uma ligação canônica   ###

 tetha <- w2%*%tanh( (w1%*%X)+(b1%*%um) ) + b2%*%um
 mi<-exp(tetha)

 ### Calcula os desvio de treinamento e Validação para a rede inicial ###

 Desvio.T[1] <- 2*sum(   ( D*log( ((D)/(mi)) + 10^-9 ) ) - (D - mi)   )

 tethav <- w2%*%tanh( (w1%*%XV)+(b1%*%umv) ) + b2%*%umv
 miv<-exp(tethav)
 Desvio.V[1] <- 2*sum(  (DV*log( (DV)/(miv) + 10^-9 )) - (DV - miv)   )
 dd <- Desvio.V[1];
```

```
print(dd)


### Calcula a norma dos pesos iniciais para a rede ###

nw1 <- as.vector(sum(w1*w1))
nw2 <- as.vector(sum(w2*w2))
nb1 <- as.vector(sum(b1*b1))
nb2 <- as.vector(sum(b2*b2))
n.w <- nw1+nw2+nb1+nb2
Norma.W[1] <- n.w


#########################################################################
###                                                                   ###
###         TREINANDO A REDE - Algoritmo BackPropagation              ###
###                                                                   ###
#########################################################################

for( k in 2:int)
{
 R.h <- tanh( (w1%*%X)+(b1%*%um) )
 R.h.deriv <- 1-R.h^2
 erro <- D - mi

 ### Calculando as Normas dos Pesos ###

 nw1 <- as.vector(sum(w1*w1))
 nw2 <- as.vector(sum(w2*w2))
 nb1 <- as.vector(sum(b1*b1))
 nb2 <- as.vector(sum(b2*b2))
 n.w <- nw1+nw2+nb1+nb2
   r <- runif(1)
 Norma.W[k] <- n.w

 ### Calculando os Gradientes ###

 grad.w1 <- ((t(w2)%*%erro)*R.h.deriv)%*%t(X)
 grad.w1 <- grad.w1/n.w

 grad.w2 <- erro%*%t(R.h)
 grad.w2 <- grad.w2/n.w

 grad.b1 <- ((t(w2)%*%erro)*R.h.deriv)%*%t(um)
 grad.b1 <- grad.b1/n.w

 grad.b2 <- erro%*%t(um)
 grad.b2 <- grad.b2/n.w
```

```
### Atualizando os pesos da Rede ###

w1 <- w1 + eta*r*grad.w1
w2 <- w2 + eta*r*grad.w2
b1 <- b1 + eta*r*grad.b1
b2 <- b2 + eta*r*grad.b2

### Calculando o Desvio de Treinamento ###

tetha <- w2%*%tanh( (w1%*%X)+(b1%*%um) ) + b2%*%um
mi<-exp(tetha)

Desvio.T[k] <- 2*sum( ( D*log( ((D)/(mi)) + 10^-9 ) ) - (D - mi) )

### Calculando o Desvio de Validação ###

tethav <- w2%*%tanh( (w1%*%XV)+(b1%*%umv) ) + b2%*%umv
miv<-exp(tethav)

Desvio.V[k] <- 2*sum( (DV*log( (DV)/(miv) + 10^-9 )) - (DV - miv) )

### Testando o Desvio de Treinamento so mantem a atualizacao se houver ###
### um decrescimento do mesmo                                          ###

if(Desvio.T[k]>Desvio.T[k-1])
{
 w1 <- w1 - eta*r*grad.w1
 w2 <- w2 - eta*r*grad.w2
 b1 <- b1 - eta*r*grad.b1
 b2 <- b2 - eta*r*grad.b2
 Desvio.T[k] <- Desvio.T[k-1]
 Desvio.V[k] <- Desvio.V[k-1]
 Norma.W[k]  <- Norma.W[k-1]
}


### Calculano os Pesos da Rede quando a Validação for mínima ###

if(Desvio.V[k] < dd)
{
 dd<-Desvio.V[k]
 pesos[[1]]<-w1;
 pesos[[2]]<-w2;
 pesos[[3]]<-b1;
 pesos[[4]]<-b2;
 pesos[[5]]<-k
 names(pesos) <- c("w1","w2","b1","b2","iteração")
 }
}
```

```
 ### Calculando o Desvio de Teste ###

 w1 <- pesos[[1]]
 w2 <- pesos[[2]]
 b1 <- pesos[[3]]
 b2 <- pesos[[4]]

 tethat <- w2%*%tanh( (w1%*%XV)+(b1%*%umt) ) + b2%*%umt
 mit<-exp(tethat)

 Desvio.Teste <- 2*sum(  (DT*log( (DT)/(mit) + 10^-9 )) - (DT - mit)   )

### Organizando a Saída da Função ###

 Result[[1]] <- Desvio.T
 Result[[2]] <- Desvio.V
 Result[[3]] <- pesos
 Result[[4]] <- Norma.W
 Result[[5]] <- Desvio.Teste
 names(Result) <- c("Desvio Treinamento","Desvio Validação","Pesos da
Rede","Norma dos Pesos","Desvio de Teste" )

 Result

}
```

Rotina de uma RNG para Binomial

```
bkp.mod.bin <- function (D,X,DV,XV,DT,XT,H,eta,int)
{

 ###   Leitura de Dados e Inicialização dos pesos da rede   ###

 X <- t(as.matrix(X)); D <- t(as.matrix(D)); n <- dim(X)[2]
 XV <- t(as.matrix(XV)); DV <- t(as.matrix(DV)); nv <- dim(XV)[2]
 XT <- t(as.matrix(XT)); DT <- t(as.matrix(DT)); nt <- dim(XT)[2]

 p <- dim(X)[1]
 w1 <- matrix((rnorm(H*p)*0.001),H,p)
 w2 <- matrix((rnorm(H)*0.001),1,H)
 b1 <- matrix((rnorm(H)*0.001),H,1)
 b2 <- matrix(log( mean(D)/(1-mean(D)) ),1,1)
 eta <- eta

 ###   Criando vetores e variáveis auxiliares   ###

 um <- matrix(1,1,n)
 umv <- matrix(1,1,nv)
 umt <- matrix(1,1,nt)
 Desvio.T <- numeric(int)
 Desvio.V <- numeric(int)
 Norma.W <- numeric(int)
 pesos <- as.list(1:5)
 Result <- as.list(1:5)

 ###   Calculando a saída da rede - supondo uma ligação canônica   ###

 tetha <- w2%*%tanh( (w1%*%X)+(b1%*%um) ) + b2%*%um
 mi<-exp(tetha)/(1+exp(tetha))

 ### Calcula os desvio de treinamento e Validação para a rede inicial ###


 parc.1 <- log( (D + 10^-44)/mi)
 parc.2 <- log( (1 - D + 10^-44)/(1 - mi))
 parc.3 <- (D*parc.1) + ( (1-D)*parc.2 )
 Desvio.T[1] <- 2*sum(parc.3)

 tethav <- w2%*%tanh( (w1%*%XV)+(b1%*%umv) ) + b2%*%umv
 miv<-exp(tethav)/(1+exp(tethav))

 parc.1 <- log( (DV + 10^-44)/miv)
 parc.2 <- log( (1 - DV + 10^-44)/(1 - miv))
 parc.3 <- (DV*parc.1) + ( (1-DV)*parc.2 )
```

```
Desvio.V[1] <- 2*sum(parc.3)

dd <- Desvio.V[1];

### Calcula a norma dos pesos iniciais para a rede ###

nw1 <- as.vector(sum(w1*w1))
nw2 <- as.vector(sum(w2*w2))
nb1 <- as.vector(sum(b1*b1))
nb2 <- as.vector(sum(b2*b2))
n.w <- nw1+nw2+nb1+nb2
Norma.W[1] <- n.w

#########################################################################
###                                                                   ###
###         TREINANDO A REDE - Algoritmo BackPropagation              ###
###                                                                   ###
#########################################################################

for( k in 2:int)
{
 R.h <- tanh( (w1%*%X)+(b1%*%um) )
 R.h.deriv <- 1-R.h^2
 erro <- D - mi

 ### Calculando as Normas dos Pesos ###

 nw1 <- as.vector(sum(w1*w1))
 nw2 <- as.vector(sum(w2*w2))
 nb1 <- as.vector(sum(b1*b1))
 nb2 <- as.vector(sum(b2*b2))
 n.w <- nw1+nw2+nb1+nb2
   r <- runif(1)
 Norma.W[k] <- n.w

 ### Calculando os Gradientes ###

 grad.w1 <- ((t(w2)%*%erro)*R.h.deriv)%*%t(X)
 grad.w1 <- grad.w1/n.w

 grad.w2 <- erro%*%t(R.h)
 grad.w2 <- grad.w2/n.w

 grad.b1 <- ((t(w2)%*%erro)*R.h.deriv)%*%t(um)
 grad.b1 <- grad.b1/n.w

 grad.b2 <- erro%*%t(um)
 grad.b2 <- grad.b2/n.w

 ### Atualizando os pesos da Rede ###
```

```
w1 <- w1 + eta*r*grad.w1
w2 <- w2 + eta*r*grad.w2
b1 <- b1 + eta*r*grad.b1
b2 <- b2 + eta*r*grad.b2

### Calculando o Desvio de Treinamento ###

tetha <- w2%*%tanh( (w1%*%X)+(b1%*%um) ) + b2%*%um
mi<-exp(tetha)/(1+exp(tetha))


parc.1 <- log( (D + 10^-44)/mi)
parc.2 <- log( (1 - D + 10^-44)/(1 - mi))
parc.3 <- (D*parc.1) + ( (1-D)*parc.2 )
Desvio.T[k] <- 2*sum(parc.3)

### Calculando o Desvio de Validação ###

tethav <- w2%*%tanh( (w1%*%XV)+(b1%*%umv) ) + b2%*%umv
miv<-exp(tethav)/(1+exp(tethav))

parc.1 <- log( (DV + 10^-44)/miv)
parc.2 <- log( (1 - DV + 10^-44)/(1 - miv))
parc.3 <- (DV*parc.1) + ( (1-DV)*parc.2 )
Desvio.V[k] <- 2*sum(parc.3)

### Testando o Desvio de Treinamento so mantem a atualizacao se houver ###
### um decrescimento do mesmo                                        ###
### print(Desvio.T[k])
if(Desvio.T[k]>Desvio.T[k-1])
{
 w1 <- w1 - eta*r*grad.w1
 w2 <- w2 - eta*r*grad.w2
 b1 <- b1 - eta*r*grad.b1
 b2 <- b2 - eta*r*grad.b2
 Desvio.T[k] <- Desvio.T[k-1]
 Desvio.V[k] <- Desvio.V[k-1]
 Norma.W[k]  <- Norma.W[k-1]
}

### Calculano os Pesos da Rede quando a Validação for mínima ###

if(Desvio.V[k] < dd)
{
 dd<-Desvio.V[k]
 pesos[[1]]<-w1;
 pesos[[2]]<-w2;
 pesos[[3]]<-b1;
 pesos[[4]]<-b2;
```

```
  pesos[[5]]<-k
  names(pesos) <- c("w1","w2","b1","b2","iteração")
 }
}

w1 <- pesos[[1]]
w2 <- pesos[[2]]
b1 <- pesos[[3]]
b2 <- pesos[[4]]

### Calculando o Desvio de Teste ###

tethat <- w2%*%tanh( (w1%*%XT)+(b1%*%umt) ) + b2%*%umt
mit<-exp(tethat)/(1+exp(tethat))

parc.1 <- log( (DT + 10^-44)/mit)
parc.2 <- log( (1 - DT + 10^-44)/(1 - mit))
parc.3 <- (DT*parc.1) + ( (1-DT)*parc.2 )
Desvio.Teste <- 2*sum(parc.3)

### Organizando a Saída da Função ###

Result[[1]] <- Desvio.T
Result[[2]] <- Desvio.V
Result[[3]] <- pesos
Result[[4]] <- Norma.W
Result[[5]] <- Desvio.Teste
names(Result) <- c("Desvio Treinamento","Desvio Validação","Pesos da
Rede","Norma dos Pesos", "Desvio de Teste")

Result

}
```

Rotina de uma RNG para Gama

```
bkp.mod.gama <- function (D,X,DV,XV,DT,XT,H,eta,int)
{

 ###   Leitura de Dados e Inicialização dos pesos da rede   ###

 X <- t(as.matrix(X)); D <- t(as.matrix(D)); n <- dim(X)[2]
 XV <- t(as.matrix(XV)); DV <- t(as.matrix(DV)); nv <- dim(XV)[2]
 XT <- t(as.matrix(XT)); DT <- t(as.matrix(DT)); nt <- dim(XT)[2]

 p <- dim(X)[1]
 w1 <- matrix((rnorm(H*p)*0.0001),H,p)
 w2 <- matrix((rnorm(H)*0.0001),1,H)
 b1 <- matrix((rnorm(H)*0.0001),H,1)
 b2 <- matrix((-n/sum(D)),1,1)
 eta <- eta

 ###   Criando vetores e variáveis auxiliares   ###

 um <- matrix(1,1,n)
 umv <- matrix(1,1,nv)
 umt <- matrix(1,1,nt)
 Desvio.T <- numeric(int)
 Desvio.V <- numeric(int)
 Norma.W <- numeric(int)

 pesos <- as.list(1:5)
 pesos[[1]]<-w1
 pesos[[2]]<-w2
 pesos[[3]]<-b1
 pesos[[4]]<-b2
 pesos[[5]]<-1
 names(pesos) <- c("w1","w2","b1","b2","iteração")

 Result <- as.list(1:5)
 ind<-numeric(n)
 indv<-indt<-numeric(nv)
 ###   Calculando a saída da rede - supondo uma ligação canônica   ###

 tetha <- w2%*%tanh( (w1%*%X)+(b1%*%um) ) + b2%*%um

 ###   Teste de Adequação de Tetha   ###

 ind = (1:n)[as.vector(tetha)>0]
 somaind = sum(ind)
 if( somaind>0 )
 {
  for(z in ind)
```

```
  tetha[z] <- -1
}

mi<- (-um/tetha)

### Calcula os desvio de treinamento e Validação para a rede inicial ###


parc.1 <- log( mi/D )
parc.2 <-  (D - mi)/mi
parc.3 <- parc.1 + parc.2
Desvio.T[1] <- 2*sum(parc.3)

tethav <- w2%*%tanh( (w1%*%XV)+(b1%*%umv) ) + b2%*%umv
###   Teste de Adequação de Tetha   ###

ind = (1:nv)[as.vector(tethav)>0]
somaind = sum(ind)
if( somaind>0 )
{
 for(z in ind)
  tethav[z] <- -1
}


miv<- (-umv/tethav)

### Calcula os desvio de treinamento e Validação para a rede inicial ###


parc.1 <- log( miv/DV )
parc.2 <-  (DV - miv)/miv
parc.3 <- parc.1 + parc.2
Desvio.V[1] <- 2*sum(parc.3)

dd <- Desvio.V[1];

### Calcula a norma dos pesos iniciais para a rede ###

nw1 <- as.vector(sum(w1*w1))
nw2 <- as.vector(sum(w2*w2))
nb1 <- as.vector(sum(b1*b1))
nb2 <- as.vector(sum(b2*b2))
n.w <- nw1+nw2+nb1+nb2
Norma.W[1] <- n.w
```

```
#########################################################################
###                                                                   ###
###         TREINANDO A REDE - Algoritmo BackPropagation              ###
###                                                                   ###
#########################################################################

 for( k in 2:int)
 {
  R.h <- tanh( (w1%*%X)+(b1%*%um) )
  R.h.deriv <- 1-R.h^2
  erro <- D - mi

  ### Calculando as Normas dos Pesos ###

  nw1 <- as.vector(sum(w1*w1))
  nw2 <- as.vector(sum(w2*w2))
  nb1 <- as.vector(sum(b1*b1))
  nb2 <- as.vector(sum(b2*b2))
  n.w <- nw1+nw2+nb1+nb2
    r <- runif(1)
  Norma.W[k] <- n.w

  ### Calculando os Gradientes ###

  grad.w1 <- ((t(w2)%*%erro)*R.h.deriv)%*%t(X)
  grad.w1 <- grad.w1/n.w

  grad.w2 <- erro%*%t(R.h)
  grad.w2 <- grad.w2/n.w

  grad.b1 <- ((t(w2)%*%erro)*R.h.deriv)%*%t(um)
  grad.b1 <- grad.b1/n.w

  grad.b2 <- erro%*%t(um)
  grad.b2 <- grad.b2/n.w

  ### Atualizando os pesos da Rede ###

  w1 <- w1 + eta*r*grad.w1
  w2 <- w2 + eta*r*grad.w2
  b1 <- b1 + eta*r*grad.b1
  b2 <- b2 + eta*r*grad.b2

  ### Calculando o Desvio de Treinamento ###

 ###   Calculando a saída da rede - supondo uma ligação canônica   ###

 tetha <- w2%*%tanh( (w1%*%X)+(b1%*%um) ) + b2%*%um
```

```
 ###   Teste de Adequação de Tetha   ###

ind = (1:n)[as.vector(tetha)>0]
somaind = sum(ind)
if( somaind>0 )
{
 for(z in ind)
  tetha[z] <- -1
}


mi<- (-um/tetha)

### Calcula os desvio de treinamento e Validação para a rede inicial ###


parc.1 <- log( mi/D )
parc.2 <-  (D - mi)/mi
parc.3 <- parc.1 + parc.2
Desvio.T[k] <- 2*sum(parc.3)

tethav <- w2%*%tanh( (w1%*%XV)+(b1%*%umv) ) + b2%*%umv
###   Teste de Adequação de Tetha   ###

ind = (1:nv)[as.vector(tethav)>0]
somaind = sum(ind)
if( somaind>0 )
{
 for(z in ind)
  tethav[z] <- -1
}



miv<- (-umv/tethav)

parc.1 <- log( miv/DV )
parc.2 <-  (DV - miv)/miv
parc.3 <- parc.1 + parc.2
Desvio.V[k] <- 2*sum(parc.3)

 ### Testando o Desvio de Treinamento so mantem a atualizacao se houver ###
 ### um decrescimento do mesmo                                          ###
 ### print(Desvio.T[k])
 if(Desvio.T[k]>Desvio.T[k-1])
 {
  w1 <- w1 - eta*r*grad.w1
  w2 <- w2 - eta*r*grad.w2
  b1 <- b1 - eta*r*grad.b1
  b2 <- b2 - eta*r*grad.b2
```

```
 Desvio.T[k] <- Desvio.T[k-1]
 Desvio.V[k] <- Desvio.V[k-1]
 Norma.W[k]  <- Norma.W[k-1]
}


### Calculano os Pesos da Rede quando a Validação for mínima ###

if(Desvio.V[k] < dd)
{
 dd<-Desvio.V[k]
 pesos[[1]]<-w1;
 pesos[[2]]<-w2;
 pesos[[3]]<-b1;
 pesos[[4]]<-b2;
 pesos[[5]]<-k
}
}

w1 <- pesos[[1]]
w2 <- pesos[[2]]
b1 <- pesos[[3]]
b2 <- pesos[[4]]

### Calculando o Desvio de Teste ###

### Calculando o Desvio de Validação ###

tethat <- w2%*%tanh( (w1%*%XT)+(b1%*%umt) ) + b2%*%umt
###   Teste de Adequação de Tetha   ###

ind = (1:nt)[as.vector(tethat)>0]
somaind = sum(ind)
if( somaind>0 )
{
 for(z in ind)
  tethat[z] <- -1
}


mit<- (-1/tethat)


parc.1 <- log( mit/DT )
parc.2 <-  (DT - mit)/mit
parc.3 <- parc.1 + parc.2
Desvio.Teste <- 2*sum(parc.3)
### Organizando a Saída da Função ###

Result[[1]] <- Desvio.T
```

```
 Result[[2]] <- Desvio.V
 Result[[3]] <- pesos
 Result[[4]] <- Norma.W
 Result[[5]] <- Desvio.Teste
 names(Result) <- c("Desvio Treinamento","Desvio Validação","Pesos da
Rede","Norma dos Pesos", "Desvio de Teste")

 Result

}
```

Rotina de uma RNG para Normal

```
bkp.mod.rna <- function (D,X,DV,XV,DT,XT,H,eta,int)
{

 ###   Leitura de Dados e Inicialização dos pesos da rede   ###

 X <- t(as.matrix(X)); D <- t(as.matrix(D)); n <- dim(X)[2]
 XV <- t(as.matrix(XV)); DV <- t(as.matrix(DV)); nv <- dim(XV)[2]
 XT <- t(as.matrix(XT)); DT <- t(as.matrix(DT)); nt <- dim(XT)[2]

 p <- dim(X)[1]
 w1 <- matrix((rnorm(H*p)*0.001),H,p)
 w2 <- matrix((rnorm(H)*0.001),1,H)
 b1 <- matrix((rnorm(H)*0.001),H,1)
 b2 <- matrix(log(mean(D)),1,1)
 eta <- eta

 ###   Criando vetores e variáveis auxiliares   ###

 um <- matrix(1,1,n)
 umv <- matrix(1,1,nv)
 umt <- matrix(1,1,nt)
 Desvio.T <- numeric(int)
 Desvio.V <- numeric(int)
 Norma.W <- numeric(int)

 pesos <- as.list(1:5)
 pesos[[1]] <- w1
 pesos[[2]] <- w2
 pesos[[3]] <- b1
 pesos[[4]] <- b2
 pesos[[5]] <- 1

 Result <- as.list(1:5)

 ###   Calculando a saída da rede - supondo uma ligação canônica   ###

 tetha <- w2%*%tanh( (w1%*%X)+(b1%*%um) ) + b2%*%um
 mi<-tetha

 ### Calcula os desvio de treinamento e Validação para a rede inicial ###

 Desvio.T[1] <- sum((D-mi)^2)
 tethav <- w2%*%tanh( (w1%*%XV)+(b1%*%umv) ) + b2%*%umv
 miv<-tethav
 Desvio.V[1] <- sum((DV-miv)^2)
 dd <- Desvio.V[1];
```

```
### Calcula a norma dos pesos iniciais para a rede ###

nw1 <- as.vector(sum(w1*w1))
nw2 <- as.vector(sum(w2*w2))
nb1 <- as.vector(sum(b1*b1))
nb2 <- as.vector(sum(b2*b2))
n.w <- nw1+nw2+nb1+nb2
Norma.W[1] <- n.w

#########################################################################
###                                                                   ###
###         TREINANDO A REDE - Algoritmo BackPropagation              ###
###                                                                   ###
#########################################################################

for( k in 2:int)
{
 R.h <- tanh( (w1%*%X)+(b1%*%um) )
 R.h.deriv <- 1-R.h^2
 erro <- D - mi

 ### Calculando as Normas dos Pesos ###

 nw1 <- as.vector(sum(w1*w1))
 nw2 <- as.vector(sum(w2*w2))
 nb1 <- as.vector(sum(b1*b1))
 nb2 <- as.vector(sum(b2*b2))
 n.w <- nw1+nw2+nb1+nb2
   r <- runif(1)
 Norma.W[k] <- n.w

 ### Calculando os Gradientes ###

 grad.w1 <- ((t(w2)%*%erro)*R.h.deriv)%*%t(X)
 grad.w1 <- grad.w1/n.w

 grad.w2 <- erro%*%t(R.h)
 grad.w2 <- grad.w2/n.w

 grad.b1 <- ((t(w2)%*%erro)*R.h.deriv)%*%t(um)
 grad.b1 <- grad.b1/n.w

 grad.b2 <- erro%*%t(um)
 grad.b2 <- grad.b2/n.w

 ### Atualizando os pesos da Rede ###

 w1 <- w1 + eta*r*grad.w1
 w2 <- w2 + eta*r*grad.w2
```

```
b1 <- b1 + eta*r*grad.b1
b2 <- b2 + eta*r*grad.b2

### Calculando o Desvio de Treinamento ###

tetha <- w2%*%tanh( (w1%*%X)+(b1%*%um) ) + b2%*%um
mi<-tetha

Desvio.T[k] <- sum((D-mi)*(D-mi))

### Calculando o Desvio de Validação ###

tethav <- w2%*%tanh( (w1%*%XV)+(b1%*%umv) ) + b2%*%umv
miv<-tethav

Desvio.V[k] <- sum((DV-miv)*(DV-miv))

### Testando o Desvio de Treinamento so mantem a atualizacao se houver ###
### um decrescimento do mesmo                                          ###
### print(Desvio.T[k])

if(Desvio.T[k] > Desvio.T[k-1])
{
 w1 <- w1 - eta*r*grad.w1
 w2 <- w2 - eta*r*grad.w2
 b1 <- b1 - eta*r*grad.b1
 b2 <- b2 - eta*r*grad.b2
 Desvio.T[k] <- Desvio.T[k-1]
 Desvio.V[k] <- Desvio.V[k-1]
 Norma.W[k]  <- Norma.W[k-1]
}


### Calculano os Pesos da Rede quando a Validação for mínima ###

if(Desvio.V[k] < dd)
{
 dd<-Desvio.V[k]
 pesos[[1]]<-w1;
 pesos[[2]]<-w2;
 pesos[[3]]<-b1;
 pesos[[4]]<-b2;
 pesos[[5]]<-k
 names(pesos) <- c("w1","w2","b1","b2","iteração")
 }
}
```

```
 ### Calculando o Desvio de Teste ###

 w1 <- pesos[[1]]
 w2 <- pesos[[2]]
 b1 <- pesos[[3]]
 b2 <- pesos[[4]]

 tethat <- w2%*%tanh( (w1%*%XV)+(b1%*%umt) ) + b2%*%umt
 mit<-tethat

 Desvio.Teste <- sum((DT-mit)*(DT-mit))

### Organizando a Saída da Função ###

 Result[[1]] <- Desvio.T
 Result[[2]] <- Desvio.V
 Result[[3]] <- pesos
 Result[[4]] <- Norma.W
 Result[[5]] <- Desvio.Teste
 names(Result) <- c("Desvio Treinamento","Desvio Validação","Pesos da
Rede","Norma dos Pesos","Desvio de Teste" )

 Result

}
```

Rotina de uma RNG para Cox

```
bkp.mod.cox <- function (X, censura, tempo, XV, censurav, tempov, XT, censurat,
tempot, H, eta, int)

{

 ###   Leitura de Dados    ###

 ordem <- order(tempo)
 X <- t(as.matrix(X))
 censura <- as.matrix(censura)
 n <- dim(X)[2]

 ordemv <- order(tempov)
 XV <- t(as.matrix(XV))
 censurav <- as.matrix(censurav)
 nv <- dim(XV)[2]

 ordemt <- order(tempot)
 XT <- t(as.matrix(XT))
 censurat <- as.matrix(censurat)
 nt <- dim(XT)[2]

 p <- dim(X)[1]

 ###    Ordenando os dados no tempo      ###

 X <- X[,ordem]
 censura <- censura[ordem,]

 XV <- XV[,ordemv]
 censurav <- censurav[ordemv,]

 XT <- XT[,ordemt]
 censurat <- censurat[ordemt,]

 ###    Inicialização dos pesos da rede     ###

 w1 <- matrix((rnorm(H*p)*0.001),H,p)
 w2 <- matrix((rnorm(H)*0.001),1,H)
 b1 <- matrix((rnorm(H)*0.001),H,1)
 b2 <- matrix((rnorm(1)*0.001),1,1)
 eta <- eta
```

```
### Criando vetores e variáveis auxiliares ###

um <- matrix(1,1,n)
umv <- matrix(1,1,nv)
umt <- matrix(1,1,nt)

logver.T <- numeric(int)
logver.V <- numeric(int)

Norma.W <- numeric(int)

pesos <- as.list(1:5)
pesos[[1]]<- w1
pesos[[2]]<- w2
pesos[[3]]<- b1
pesos[[4]]<- b2
pesos[[5]]<- 1

grad.w1<-0 * w1
grad.b1<-0 * b1
Result <- as.list(1:4)

T <- matrix(1,n,n)
for (l in 1:(n-1))
 for(c in 1:n)
  if(l<c)
   T[l,c] <- 0

TV <- matrix(1,nv,nv)
for (l in 1:(nv-1))
 for(c in 1:nv)
  if(l<c)
   TV[l,c] <- 0

TT <- matrix(1,nt,nt)
for (l in 1:(nt-1))
 for(c in 1:nt)
  if(l<c)
   TT[l,c] <- 0

### Calculando a saída da rede e a log-verossimilhança para as amostras de
treinamento e validação ###

tetha <- w2%*%tanh( (w1%*%X)+(b1%*%um) ) + b2%*%um
logver.T[1] <-t(censura)%*%t(tetha - t(log(T%*%exp(t(tetha)))))

tethav <- w2%*%tanh( (w1%*%XV)+(b1%*%umv) ) + b2%*%umv
logver.V[1] <-t(censurav)%*%t(tethav - t(log(TV%*%exp(t(tethav)))))

dd <- logver.V[1];
```

```
### Calcula a norma dos pesos iniciais para a rede ###

nw1 <- as.vector(sum(w1*w1))
nw2 <- as.vector(sum(w2*w2))
nb1 <- as.vector(sum(b1*b1))
nb2 <- as.vector(sum(b2*b2))
n.w <- nw1+nw2+nb1+nb2
Norma.W[1] <- n.w

#########################################################################
###                                                                   ###
###         TREINANDO A REDE - Algoritmo BackPropagation              ###
###                                                                   ###
#########################################################################

for( k in 2:int)
{

 ### Calculando as Normas dos Pesos ###

 nw1 <- as.vector(sum(w1*w1))
 nw2 <- as.vector(sum(w2*w2))
 nb1 <- as.vector(sum(b1*b1))
 nb2 <- as.vector(sum(b2*b2))
 n.w <- nw1+nw2+nb1+nb2
   r <- runif(1)
 Norma.W[k] <- n.w

 ### Calculando os Gradientes ###
 ### Os Gradientes dos pesos da camada de saída são calculados vetorialmente  ###
 ### Enquanto que os Gradientes dos pesos da camada escondida são calculados por
etapa, ou seja, nodo a nodo ###

 V <- tanh((w1%*%X)+(b1%*%um))
 dV <- 1 - V*V
 tetha <- (w2%*%V) + (b2%*%um)

 grad.w2 <- t(t( t(V) - diag(as.vector(t(um) / (T %*% exp(t(tetha))))) %*% T %*%
(diag(as.vector(exp(t(tetha)))) %*% t(V))) %*% censura)

 grad.b2 <- t(t( t(um) - diag(as.vector(t(um) / (T %*% exp(t(tetha))))) %*% T %*%
(diag(as.vector(exp(t(tetha)))) %*% t(um))) %*% censura)

 ### Gradientes dos pesos da camada escondida ###

 for (h in 1: H)
 {
  aux.um <- w2[1,h,drop=F] %*% dV[h,,drop=F]
     aux <- matrix(rep(as.vector(aux.um),p),nrow=p,byrow=T)
```

```
   aux.V <-aux*X


  grad.w1[h,] <- t(t( t(aux.V) - diag(as.vector(t(um) / (T %*% exp(t(tetha))))) %*% T
%*% (diag(as.vector(exp(t(tetha)))) %*% t(aux.V))) %*% censura)

  grad.b1[h,1] <- t(t( t(aux.um) - diag(as.vector(t(um) / (T %*% exp(t(tetha))))) %*% T
%*% (diag(as.vector(exp(t(tetha)))) %*% t(aux.um))) %*% censura)
 }

 ### Normalização dos vetores Gradiente ###

 grad.w2 <- grad.w2/n.w
 grad.b2 <- grad.b2/n.w
 grad.w1 <- grad.w1/n.w
 grad.b1 <- grad.b1/n.w

 ### Atualizando os pesos da Rede ###

 w1 <- w1 + eta*r*grad.w1
 w2 <- w2 + eta*r*grad.w2
 b1 <- b1 + eta*r*grad.b1
 b2 <- b2 + eta*r*grad.b2

 ### Calculando as log-verossimilhanças de Treinamento e Validação###

 tetha <- w2%*%tanh( (w1%*%X)+(b1%*%um) ) + b2%*%um
 logver.T[k] <-t(censura)%*%t(tetha - t(log(T%*%exp(t(tetha)))))

 tethav <- w2%*%tanh( (w1%*%XV)+(b1%*%umv) ) + b2%*%umv
 logver.V[k] <-t(censurav)%*%t(tethav - t(log(TV%*%exp(t(tethav)))))

 ### Testando o Desvio de Treinamento so mantem a atualizacao se houver ###
 ### um decrescimento do mesmo                                        ###
 ### print(logver.T[k])

 if(logver.T[k]<logver.T[k-1])
 {
  w1 <- w1 - eta*r*grad.w1
  w2 <- w2 - eta*r*grad.w2
  b1 <- b1 - eta*r*grad.b1
  b2 <- b2 - eta*r*grad.b2
  logver.T[k] <- logver.T[k-1]
  logver.V[k] <- logver.V[k-1]
  Norma.W[k]  <- Norma.W[k-1]
 }
```

```
 ### Calculano os Pesos da Rede quando a Validação for mínima ###

 if(logver.V[k] > dd)
 {
  dd<-logver.V[k]
  pesos[[1]]<-w1;
  pesos[[2]]<-w2;
  pesos[[3]]<-b1;
  pesos[[4]]<-b2;
  pesos[[5]]<-k
  names(pesos) <- c("w1","w2","b1","b2","iteração")
 }
}

### Calculando as log-verossimilhanças da amostra de Teste ###

w1 <- pesos[[1]]
w2 <- pesos[[2]]
b1 <- pesos[[3]]
b2 <- pesos[[4]]

tethat <- w2%*%tanh( (w1%*%XT)+(b1%*%umt) ) + b2%*%umt
logver.Teste <-t(censurat)%*%t(tethat - t(log(TT%*%exp(t(tethat)))))

### Organizando a Saída da Função ###

Result[[1]] <- logver.T
Result[[2]] <- logver.V
Result[[3]] <- pesos
Result[[4]] <- Norma.W
Result[[5]] <- logver.Teste
names(Result) <- c("log-verossimilhaça Treinamento","log-verossimilhança
Validação","Pesos da Rede","Norma dos Pesos","log-verossimilhaça Teste")

Result

}
```